IBM

Personal Computer
Language Series

BASIC Programming Development System

- Enhanced Editor
- Structured BASIC Preprocessor
- BASIC Formatter
- BASIC Cross-Reference

Continued on inside back cover

Personal Computer
Language Series

BASIC Programming Development System

- Text File Editor
- Structured BASIC Preprocessor
- BASIC Formatter
- BASIC Cross Reference

First Edition (November 1982)

Preface

This book teaches you how to use the four utilities in the IBM Personal Computer BASIC Programming Development System package: the Text File Editor, the Structured BASIC Preprocessor, the BASIC Formatter, and the BASIC Cross-Reference.

This book gives you the following information:

- What each utility does and how it works
- The advantages of each utility
- Practice examples demonstrating the utilities
- Explanations of the commands and the functions of each utility

Assumptions

This book assumes that you are familiar with programming concepts in general and BASIC programming in particular. This book is not designed to teach you programming; rather, its purpose is to provide you with reference information for a powerful program development tool.

In addition, this book assumes that you have read IBM Personal Computer *Guide to Operations,* IBM Personal Computer *Disk Operating System* (DOS), and IBM Personal Computer *BASIC.*

You need to be familiar with the IBM Personal Computer and the DOS commands to use the utilities in this package.

Book Structure

The IBM Personal Computer *BASIC Programming Development System* book is divided into five chapters and six appendixes:

Chapter 1 contains an introduction to the BASIC Programming Development System package and gives you an overview of each utility. This chapter lists the characteristics of each utility, explains how each utility operates, lists the program requirements, and gives you the important facts you need to know to operate the program and the utilities.

Chapter 2 explains how to load the BASIC Programming Development System program, how to copy DOS, and how to make a backup of the program diskette.

Chapter 3 explains structured BASIC and shows you the design and the syntax of the structured macros. This discussion includes macro flowcharts that show you how to design your own BASIC programs using structured BASIC macros.

Chapter 4 is a tutorial that teaches you how to use the major commands of the four utilities in the BASIC Programming Development System.

Chapter 5 is a complete reference section that lists the BASIC Programming Development System commands and details their purpose, function, and use.

Appendix A provides you with a complete listing of the BASIC Programming Development System error messages.

Appendix B shows you how to convert a file into ASCII format.

Appendix C explains how to convert an existing file into structured BASIC.

Appendix D examines two EXEC files, DEBUG and FREQ, that offer you specialized ways of preprocessing files.

Appendix E gives you directions for editing and preprocessing large files.

Appendix F shows you how to edit an EXEC file, AUTOEXEC. TFE.

Special Hints

As you progress through this book, set a reasonable pace for yourself based on your level of expertise. The four utilities in the BASIC Programming Development System offer you a wide variety of commands and functions that you can use to fulfill your own programming needs. We recommend that you read the first four chapters of this book carefully and then experiment with the commands listed in the reference section.

To help you learn the utilities in this package, we've designed this book so you can see at a glance what you need to type and what appears on the screen. The text you type and enter is in **boldface**, and the sample screens appear in `green`

Contents

Chapter 1. Introduction

Contents

The Utilities

The BASIC Programming Development System package is a program development tool. It contains four utilities: the Text File Editor, the Structured BASIC Preprocessor, the BASIC Formatter, and the BASIC Cross-Reference.

Let's examine some special characteristics of these utilities.

Text File Editor

The Text File Editor (TFE) helps you create and edit programs using both line and full-screen editing capabilities. This utility works in conjunction with the Structured BASIC Preprocessor.

Characteristics of the Text File Editor:

- Contains a full range of line editing commands
- Offers limited full-screen editing
- Provides on-line help for the commands
- Offers full use of the 10 function keys
- Contains formatting commands for word processing
- Offers printing and filing capabilities
- Stamps date and time on printouts
- Includes paging control for LIST, PRINT, FIND, and TYPE
- Enables you to create DOS files in ASCII format

Structured BASIC Preprocessor

The Structured BASIC Preprocessor (SBP) enables you to process structured BASIC source files.

Characteristics of the Structured BASIC Preprocessor:

- Expands structured macros into actual BASIC statements
- Converts labels into line numbers
- Automatically combines lines and removes remarks to save storage space
- Enables you to preprocess existing unstructured BASIC programs

BASIC Formatter and Cross-Reference

The BASIC Formatter and Cross-Reference (FRMTBAS) work together on ASCII files to produce formatted and cross-referenced listings of unstructured BASIC source programs. These utilities are options on the same menus, but you can use them separately.

Characteristics of the BASIC Formatter and Cross-Reference:

- Separates and indents FOR-NEXT, IF-THEN, and WHILE-WEND statements to show a program's structure
- Separates and indents DATA, DIM, and FIELD statements to enhance readability
- Indents all code so line numbers remain visible, with continuation lines indented past the line numbers
- Breaks multiple statements into single lines
- Places blank lines before and after remarks

- Offers a cross-reference option for statement numbers and shows the lines referencing those numbers
- Offers a cross-reference option for variables and shows the lines referencing those variables
- Offers a cross-reference option for reserved words and shows the lines referencing those words

What You Need

The BASIC Programming Development system has the following minimum requirements:

- IBM Personal Computer
- One diskette drive required; two diskette drives preferred
- 96KB of memory
- IBM Monochrome Display or any other video display screen that is compatible with the IBM Personal Computer
- IBM Personal Computer Disk Operating System (DOS)
- IBM 80 CPS Matrix Printer or any other compatible printer

Things to Remember

The BASIC Programming Development System consists of two programs. The one program contains the Text File Editor and the Structured BASIC Preprocessor. The other program contains the BASIC Formatter and the BASIC Cross-Reference.

In this book, we frequently refer to these programs and the utilities on them as *TFE/SBP* and *FRMTBAS*. Keep these abbreviations in mind as you progress through the book.

You can use these utilities in any order that suits your needs. To do this, though, you have to know some facts about the operation of each utility and the programs that contain them.

For this reason, we've collected all of the important facts, restrictions, and limitations you need to know in this section. Although some of this information may not mean that much to you right now, you should refer back to this section whenever you use any of the utilities.

The specific items on this list are discussed in more detail in the appropriate sections of the book. For the moment, study these facts to become familiar with the operation of the utilities.

General Facts

- The utilities accept commands in either uppercase or lowercase.
- The drive with DOS in it is always the default drive, unless you decide to change it.

- All printouts contain date and time stamps.

- We follow the DOS command parameter definitions in this book. A *filespec* includes the device (*d*:), the *filename* (first 1-8 characters), and the filename extension (*.ext*).

TFE/SBP

- The Text File Editor and the Structured BASIC Preprocessor (TFE/SBP) set aside a part of memory on which you perform editing and structured BASIC commands. This area is called the *edit buffer*. The edit buffer accepts up to 999 lines, using up to approximately 13KB of storage in a 96KB machine or 45KB in a 128KB machine.

- TFE/SBP uses editor line numbers. These numbers change as you add, modify, and delete lines, but they have no affect on the contents of your text files.

- TFE/SBP commands have from zero to five parameters that you separate by blanks. These parameters determine the operation of each command.

- The first two parameters usually specify the first and last lines of the file on which the command operates. The line range is from 1 to 999.

- If you specify only the first line parameter, you only receive the line you specified (the last parameter defaults to the first line parameter you specified). If you don't specify either the first or the last line you want, the parameters default to the first and last lines of the file. If you use the "at" sign (@) or 0 as the last line parameter, this line defaults to the last line of the file.

- TFE/SBP commands don't require double quotes (") on filespecs.
- Structures may not be nested more than 10 levels deep.
- Source files may not contain more than 250 labels, coded or generated.
- SBP doesn't support a question mark (?) as a substitute for the PRINT keyword.

FRMTBAS

- The BASIC Formatter and Cross-Reference (FRMTBAS) only work on files in ASCII format.
- You cannot use FRMTBAS on structured BASIC programs. FRMTBAS does work on *preprocessed* structured programs.

Chapter 2. Getting Started

Contents

Creating a Working Diskette

This section shows you how to make a backup copy of your program diskette. Before you begin, turn on the computer, load DOS, and format a backup diskette to include the system files (FORMAT [*d*:]/S).

If you have difficulty following the directions, refer to your *DOS* manual.

> **Note:** The copy you make may be used for your own purposes only; any other uses violates copyright law.

Follow these instructions:

1. Insert the BASIC Programming Development System program diskette in drive A.
2. If you have a two-drive system, insert the formatted diskette in drive B. If you have a one-drive system, you will be prompted to insert the formatted diskette later with the message **Insert diskette for drive B: and strike any key when ready**.
3. With the program diskette in drive A, type **SETUP** and press Enter.
4. On a two-drive system, the program files are automatically copied. On a one-drive system, you have to swap diskettes as prompted until all of the program files are copied.

5. After the program files have been copied to the backup diskette, the following prompt appears on your screen:

```
Pause. Insert DOS diskette in drive A:
Strike a key when ready . . .
```

6. Insert the DOS diskette in drive A and press any key.

   BASIC.COM and TIME.COM are copied to the working diskette. Again, for a one-drive system, you are prompted to insert the correct diskette. The drive A diskette is the DOS diskette, and the drive B diskette is the working diskette.

7. When the DOS prompt, **A>**, reappears, you've successfully created a working diskette. Do not be concerned if the last message you receive is **0 files copied**.

8. Store your original BASIC Programming Development System diskette in a safe place.

Optional Procedures

Moving Files to a Formatted Diskette

If you have a one-drive system, you may want to move some of the files on the BASIC Programming Development System diskette to another diskette.

If you want to move files, use the DOS DISKCOPY command and then erase the unwanted files from this new diskette.

We recommend that you move the BASIC Formatter and Cross-Reference utilities to another diskette. This file is on the program diskette as FRMTBAS.

To do this, make a copy of your backup diskette and erase the FRMTBAS file from one diskette and the TFE files from the other diskette.

Entering the Time

Since any printouts you receive with the utilities in this program include a time stamp, you probably want to enter the time whenever you use the program. Depending on which version of DOS you have, you either are prompted for the time on the first screen, or you have to enter the time by using the DOS TIME command.

If you are using the TIME command, type **TIME** and press the Enter key.

The following prompt appears on your screen:

```
Current time is hh:mm:ss.xx
Enter new time:
```

In case you are unfamiliar with this format, *hh* is the hour (0-23), *mm* is the minutes (0-59), and *ss.xx* is the seconds (0-59) and hundredths of seconds (0-99). Make your entry in military time.

If you are prompted for the time on the first screen, also be certain to use military time.

Chapter 3. Structured BASIC Concepts

Contents

General Information

Although IBM Personal Computer BASIC uses FOR-NEXT, IF-THEN, and WHILE-WEND to implement simple structures, it is not fully structured code. A programmer, especially one who did not write the code, may have difficulty looking at a program and seeing where a block of code is entered and exited and discovering the flow of control within the block of code. Structured BASIC alleviates this problem by making BASIC code easy to read and to understand.

With structured BASIC code, you can show the relationships between the parts of your program. By illustrating these internal relationships, you'll find it easier to go back into your code and make fixes, adaptations, and modifications.

A fundamental concept of structured BASIC is that the more obvious the flow of control of the program, the higher the reliability and maintainability of the code.

The flow of control in a structured BASIC program is implemented by structures. A *structure* is a block of code or a group of macros. A *macro* is a single instruction within a structure. In structured BASIC, the macros expand into the BASIC statements required to implement each structure.

It is easy to see the flow of control within structures since they always are entered at the top and exited at the bottom. This simplifies the search for the flow of control. It always flows in a top-down direction.

Beyond increasing readability, structured BASIC offers other advantages.

Structured BASIC eliminates the need for line numbers by using labels instead. These labels are strings of non-blank characters that begin with the "at" sign (@). Since all labels begin with the at sign, they are easy to find.

Furthermore, in structured BASIC, the only source lines that ever are called by a GOSUB, or even a GOTO, are lines that begin with labels.

Beyond making lines easy to identify, structured BASIC further simplifies programs by providing the ability to have all flow of control handled by macros and BASIC commands.

Now that you understand some of the advantages of structured BASIC, let's examine the structured macros.

Structured Macros

In this section, we give you an overview of structured macros. To understand the information, you need to have a general understanding of structured code and flowcharts, along with a background in BASIC programming.

There are three main structures: the IF structure, the DO structure, and the SEARCH structure.

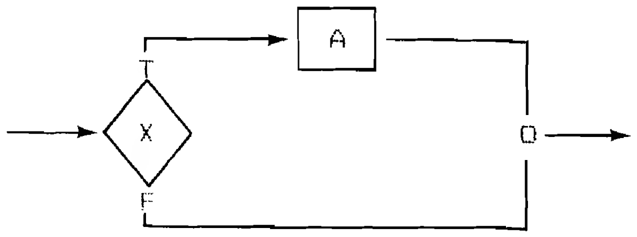
Let's look at these structures to see what they do and how they operate.

The IF Structure

You use the IF structure whenever you want to choose among a number of possible options. The IF structure is coded using IF, ELSEIF, ELSE, and ENDIF macros.

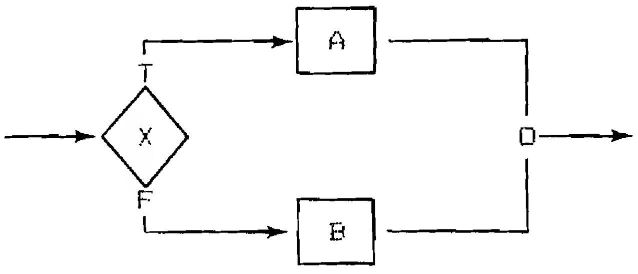
The IF structure has three standard forms — the simple IF without ELSE, the simple IF, and the cascaded IF. The IF flowcharts and their Structured BASIC and BASIC equivalents follow.

Simple IF without ELSE:



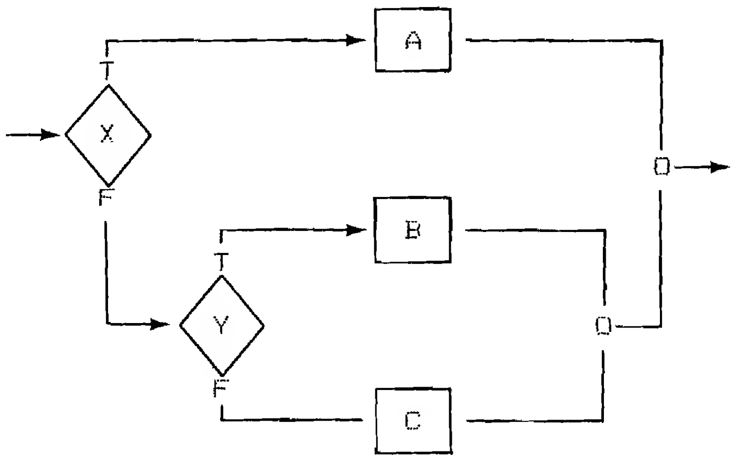
```
IF X
  A                      10 IF X THEN A
ENDIF
```

Simple IF:



```
IF X
   A
ELSE              10 IF X THEN A ELSE B
   B
ENDIF
```

Cascaded IF:



```
IF X
   A
ELSEIF Y
   B
ELSE
   C
ENDIF
```

```
10 IF X THEN A ELSE IF Y THEN B ELSE C
```

One of the major advantages of the IF structure over the IF statement is that the number of BASIC statements on each "leg" of the IF structure is not limited to the number that will fit on one line, as it is in normal BASIC. Instead, the only limit is the size of the edit buffer.

The macros of the IF structure have the following syntax:

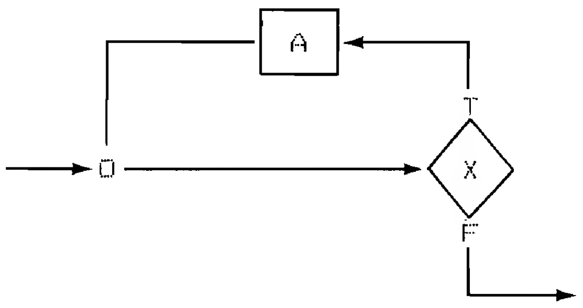
IF *condition*
ELSEIF *condition*
ELSE
ENDIF

The DO Structure

The DO structure replaces the different types of loops used in BASIC. It is coded using the DO, LEAVE, and ENDDO macros.

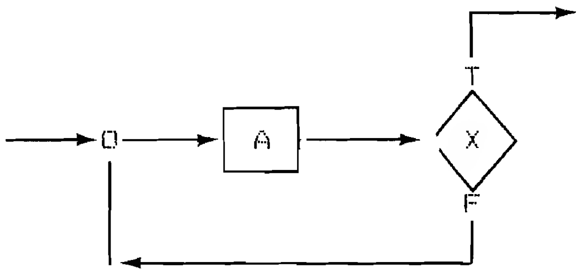
The DO structure comes in three varieties — the leading test (DO-WHILE), the trailing test (DO-UNTIL), and the middle test (DO-COMPLEX). These loops and their BASIC equivalents follow.

DO WHILE:



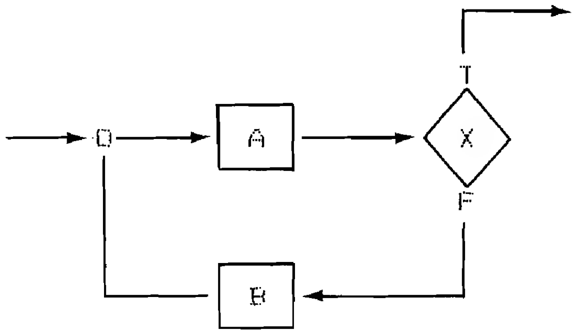
```
DO X                      10 WHILE X
  A                       20 A
ENDDO                     30 WEND
```

DO UNTIL:



```
DO                     10 REM
  A                    20 A
ENDDO X                30 IF X=0 THEN 10
```

Complex DO:



```
DO                     10 REM
  A                    20 A
LEAVE X                30 IF X THEN 60
  B                    40 B
ENDDO                  50 GOTO 10
                       60 REM
```

DO structures have two more options that are not obvious from the examples. Since the DO-UNTIL form requires a dummy variable to implement the WHILE-WEND loop, it uses the variables Q0 to Q9. The numbers 0 through 9 correspond to the level of nesting or number of other structures within the structure. If this loop is coded in a subroutine with the same variable as in the main progam, errors can occur. To avoid this, you may specify the variable to be used instead of Q by entering:

```
DO @I
```

This specifies I as the loop variable.

To generate a true FOR-NEXT loop on a given variable, you may specify exactly the same parameters on the DO structure that you would on the FOR statement.

For example, the following two loops perform the same function:

```
DO I = 1 TO 5 STEP 2       10 FOR I = 1 TO 5 STEP 2
ENDDO                      20 NEXT
```

One advantage of the DO loop over the FOR-NEXT loop is the LEAVE structure. It generates a loop exit without you having to worry about labels. Another advantage is that the DO macro is recognized by the INDENT command.

The macros of the DO structure have the following syntax:

DO [*condition/@variable*]
LEAVE *condition*
ENDO [*condition*]

The square brackets, [ ], represent options.

The SEARCH Structure

The SEARCH structure is a combination of the DO and the IF structures. You use the SEARCH structure whenever you want a loop to search for something and different processing is required for the success or the failure of the search. The SEARCH structure is coded using the SEARCH, EXITIF, ORELSE, ENDLOOP, and ENDSRCH macros.

The following lists compare SEARCH with DO/IF.

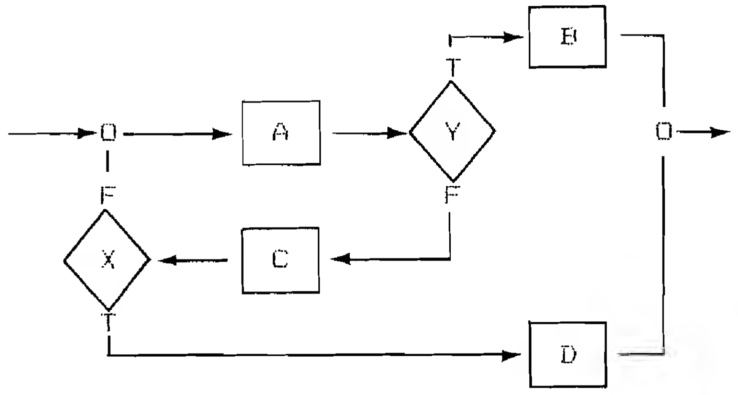
```
SEARCH x                DO x
   initial code              initial code
EXITIF y                     Z = y
   success code         LEAVE Z
ORELSE                       increment code
   increment code       ENDDO
ENDLOOP                 IF Z
   failure code              success code
ENDSRCH                 ELSE
                             failure code
                        ENDIF
```

You code the corresponding macros in exactly the same way:

SEARCH	DO
EXITIF	LEAVE
ORELSE	ELSE
ENDLOOP	ENDDO
ENDSRCH	ENDIF

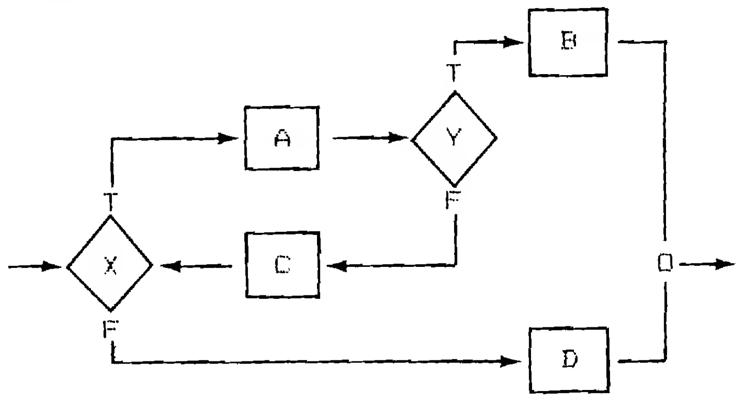
The flowcharts and coding examples for the SEARCH structure follow.

SEARCH UNTIL:



```
SEARCH
   A
EXITIF Y
   B
ORELSE
   C
ENDLOOP X
   D
ENDSRCH
```

SEARCH WHILE:



```
SEARCH X
   A
EXITIF Y
   B
ORELSE
   C
ENDLOOP
   D
ENDSRCH
```

The macros of the SEARCH structure have the following syntax:

SEARCH [*condition/@variable*]
EXITIF *condition*
ORELSE
ENDLOOP [*condition*]
ENDSRCH

The square brackets, [ ], represent options.

Now that you have seen the macros used in structured BASIC, let's examine some conventions of structured BASIC.

Structured BASIC Conventions

Structured BASIC has some syntax and coding conventions you need to know:

- You may code the macros in any combination of uppercase and/or lowercase characters. If you code parameters or remarks on the macro, you must place at least one blank after the macro name.
- Two single quotes (' ') indicate code that is to be included during preprocessing in debug mode via EX DEBUG.
- Labels are strings of non-blank characters at the beginning of lines that begin with an at sign (@) and are followed by a blank. A label may not contain a comma, a colon, a single quote, or a minus sign. There are no limits on string length.
- You code labels in exactly the same manner as line numbers.
- Line numbers in the preprocessed BASIC program begin at 1 and increment by 1, unless explicitly changed. You do this by specifying a line number before anything else on a line. Thereafter, all line numbers begin with that number and increment by 1. This is used to synchronize the generated line numbers in each source file of a multiple file program. These files later may be merged into one large program using the BASIC MERGE command. It also is helpful to establish specific line numbers for debugging.

- All structured BASIC macros must be on lines by themselves. The following is a list of all the macros:

IF	DO	SEARCH
ELSEIF	LEAVE	EXITIF
ELSE	ENDDO	ORELSE
ENDIF		ENDLOOP
		ENDSRCH

Chapter 4. Tutorial

Contents

This chapter teaches you how to use the utilities in the BASIC Programming Development System package. The material is divided into three lessons: editing with the Text File Editor, preprocessing with the Structured BASIC Preprocessor, and using the BASIC Formatter and Cross-Reference.

The lessons in this tutorial build on each other, so follow them in order. First you'll write and edit a file, then you'll preprocess that file, and finally, you'll obtain formatted and cross-referenced listings of this file.

> **Note:** The lessons are designed for a two-drive system. If you have a one-drive system, you have to make the necessary changes. For example, if we tell you to use drive B, you have to use drive A.

Lesson 1. Using TFE

Loading TFE

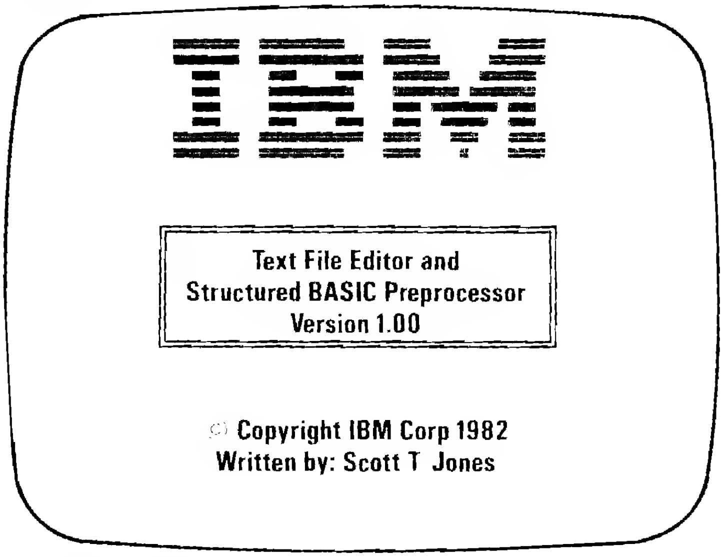
The Text File Editor (TFE) is contained in a file named TFE.EXE on the BASIC Programming Development System diskette. You load this file from DOS. If you followed the instructions in Chapter 2, DOS should be on your working diskette (from now on, this diskette is referred to as the program diskette). If you didn't, return to Chapter 2 and do so now.

In addition, if you moved the TFE files to a separate diskette, be certain you are using the correct diskette. You can verify this by using the DOS DIR command.

Load DOS. After you enter the date and the time, the DOS prompt (**A >**) appears on your screen.

Type **TFE** after the prompt and press Enter.

The following copyright message appears on the screen:



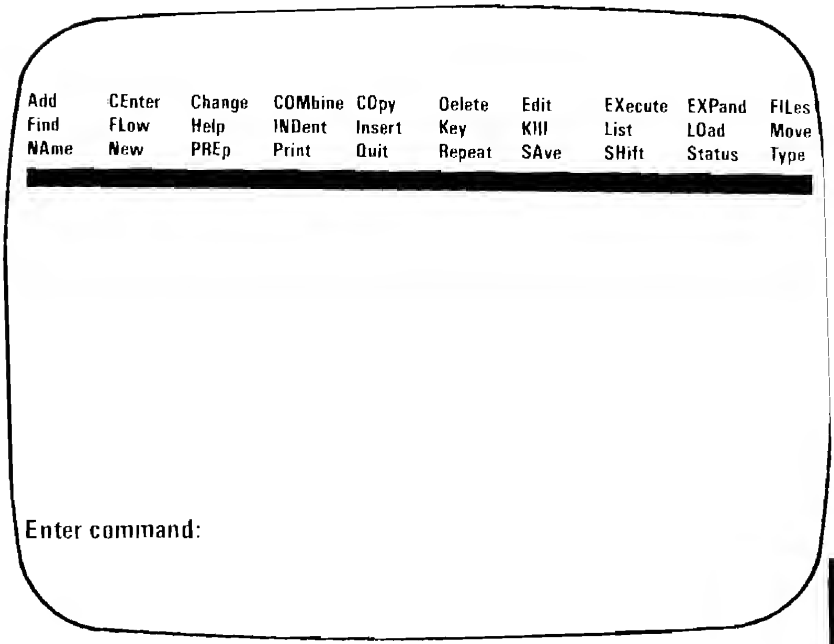
```
Text File Editor and
Structured BASIC Preprocessor
Version 1.00

© Copyright IBM Corp 1982
Written by: Scott T Jones
```

This copyright notice remains on the screen until you press the Enter key. Do so now.

Although you can't see it, the Text File Editor now loads the function key definitions from AUTOEXEC.TFE, and exec file we'll look at later.

Right now, examine your first screen. It looks like this:



```
Add     CEnter  Change  COMbine COpy    Delete  Edit    EXecute EXPand  FILes
Find    FLow    Help    INDent  Insert  Key     KIll    List    LOad    Move
NAme    New     PREp    Print   Quit    Repeat  SAve    SHift   Status  Type
```

```
Enter command:
```

A list of the commands appears at the top of the screen. This is the *command display*, and it appears anytime you enter an incorrect command name or abbreviation. You can make the display appear and disappear by pressing Enter without typing a command.

The capital letters in each command name show the minimum command abbreviations. For example, you can abbreviate ADD simply by entering the letter A, but to abbreviate CENTER, you must enter at least the letters C and E.

Study the commands and their abbreviations for a moment. As you'll discover, the commands you'll use most often have the shortest abbreviations.

At the bottom of the screen, this prompt appears:

Enter command:

This is the *command prompt.* You enter the commands and their parameters on this line.

In the next section, we'll look at some of the TFE commands you'll use frequently.

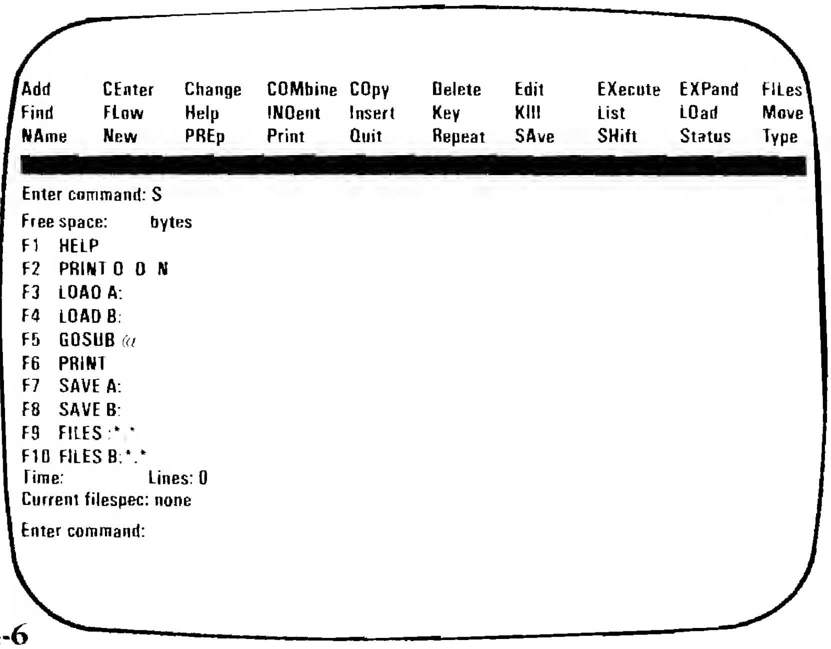
Displaying the Status of a File

One of the first commands you need to be familiar with is the STATUS command. This command gives you a mini-report on the file you are editing.

Let's take a look at the type of material the STATUS command gives you.

After the command prompt, type S and press the Enter key. (As the command display shows, the letter S is the abbreviation for the STATUS command.)

The STATUS screen has this form:



```
Add     CEnter  Change  COMbine COpy    Delete  Edit    EXecute EXPand  FILes
Find    FLow    Help    INDent  Insert  Key     Kill    List    LOad    Move
NAme    New     PREp    Print   Quit    Repeat  SAve    SHift   Status  Type

Enter command: S
Free space:      bytes
F1   HELP
F2   PRINT O  O  N
F3   LOAD A:
F4   LOAD B:
F5   GOSUB @
F6   PRINT
F7   SAVE A:
F8   SAVE B:
F9   FILES :*.*
F10  FILES B:*.*
Time:             Lines: 0
Current filespec: none
Enter command:
```

The numbers don't appear in our example because the ones you have may be different (depending on your memory size, the time of day, etc).

The STATUS screen displays the following information:

- The amount of free space left in the edit buffer
- The function key definitions
- The time
- The number of lines in the edit buffer
- The name of the file in the edit buffer
- The command prompt

You can use the STATUS command on any file.

Another feature of the STATUS command is that it clears the screen if you enter CLS as its parameter. By specifying CLS, only the command display and the command prompt appear on the screen.

Using the Function Keys

As we previously mentioned, the program automatically assigns frequently-used commands to the functions keys when you load the program. Since the function keys are "soft" keys (you can assign different definitions to them), you may change the meaning of each key with the KEY command. The KEY command is explained in detail in Chapter 5, "Reference."

Let's look at the function keys. If you don't still have the STATUS screen displayed, type **S** and press Enter.

The keys have the following definitions:

```
F1  HELP
F2  PRINT 0 0 N
F3  LOAD A:
F4  LOAD B:
F5  GOSUB @
F6  PRINT
F7  SAVE A:
F8  SAVE B:
F9  FILES A:*.*
F10 FILES B:*.*
```

F1 (HELP), F9 (FILES A:*.*), and F10 (FILES B:*.*) are "hot" keys. By "hot," we mean that you don't have to press Enter to execute these commands. These keys execute automatically.

The other keys simply give you part of a command. You have to supply the parameters or press Enter to execute the command.

Using the HELP Command

TFE makes editing easy for you in several ways. Besides the command display, you can get detailed information about a specific command by typing H or HELP and the name or abbreviation of the command. The purpose and the parameters for that command appear on the screen.

In addition, you can get HELP information by simply pressing the F1 function key.

You also can get help information on the structured macros you use with the Structured BASIC Preprocessor. This is the only HELP file that is not related to a command. For this reason, the macro abbreviation is not listed on the command display. To get the macro HELP file, you type *H MACROS* or *H MA*.

Let's look at the HELP file.

Type **H** and press Enter.

The HELP screen looks like this:

```
The HELP command displays information
about the commands. You may obtain
specific help for each command displayed
at the top of the screen. In addition,
you may obtain help on the macro syntax
by specifying H MAcros.

HELP commandname

commandname = name or abbreviation
              of the command for which
              help is desired (default
              is HELP)
```

This file explains the purpose and syntax for the HELP command. The HELP files explain every command in this same format. Look at some of the other HELP files before you go on. These files offer you quick reference for command information.

Ending the Program

Another command you'll use frequently is the QUIT command. The QUIT command ends TFE and returns you to DOS

Don't do this now because it cancels TFE. Just read the instructions on how to use the QUIT command.

To end the program, you type **Q** (the abbreviation for QUIT) after the command prompt and press Enter.

Since this command causes you to lose changes you made in the edit buffer (unless you save the changes first), you have to verify your decision to end the program.

This message appears on your screen:

```
QUIT (Y/N)?
```

If you're certain you want to end TFE, type **Y** for yes, and press Enter. If you don't want to end the program, type **N** for no, and the QUIT command is cancelled. The command prompt then returns.

When you answer Y, you receive this message:

```
TFE/SBP Ended
```

The DOS prompt appears directly below this message.

Creating a File

Now that you're familiar with the HELP, STATUS, and QUIT commands, you're ready to write and edit a text file.

Clearing the Buffer

When you begin a file, you first use the NEW command to clear the edit buffer. If the edit buffer is empty, the command prompt returns immediately. If, however, a file is in the edit buffer, you receive the prompt, **NEW (Y/N)?**, where *Yes* deletes the lines in the edit buffer and *No* cancels the NEW command.

Try the NEW command. Type **N** (the abbreviation for NEW) and press Enter.

Since the edit buffer is empty, the command prompt returns immediately. You're ready to enter your program.

Entering Lines

You use either the EDIT command or the INSERT command to create a file.

The EDIT command places you in full-screen edit mode, where you add lines by typing over data. When the edit buffer is empty, you simply enter your lines.

The INSERT command places new lines in the edit buffer. When the buffer is empty, you enter your text beginning with line 001. A new line number appears every time you press Enter after a line of text. The INSERT command cancels when you press Enter after a null (blank) line.

Let's enter a sample program using the EDIT command. After the command prompt, type **E** and press Enter.

The screen clears, and after a brief pause, a blinking cursor appears in the upper lefthand corner of the screen. This position is line 1, column 1. With the EDIT command, you can edit 24 lines at a time, with a maximum length of 255 characters per line.

At the bottom of the screen (line 25), a highlighted message appears. This message tells you how to exit the full-screen edit mode.

If you press the Ctrl and Home keys simultaneously, you exit full-screen mode and save the results of your editing to the edit buffer. When you use Ctrl-Home, the message **EDIT n saved** appears on your screen. The *n* stands for the first line you edited.

If you press the Ctrl and PgDn keys simultaneously, you exit full-screen mode without saving new material or changes. In this case, the message **EDIT n cancelled** appears on your screen.

An important point to remember is that the EDIT command only works on 24 lines at a time. Therefore, you need to save the text after you finish a screen.

Sample Program

You're ready to type and enter the sample program. This sample program is a game and uses structured BASIC. As you enter the lines, notice how the various macros are used.

Remember to save the sample program after you reach the end of line 24. Use the Ctrl-Home keys. You'll receive the message **EDIT 1 saved**. Then, type **E 20** and press Enter. This will allow you to continue entering lines, and lines 20 through 24 will be displayed so you can see where you left off.

Type and enter the following lines:

```
KEY OFF:CLS'Number Guessing Game
RANDOMIZE VAL(RIGHT$(TIME$,2))
I=0:DIM S$(5)
DO
I=I+1
LEAVE I>5
READ S$(I)
ENDDO I=0
PRINT "I've got a number from 1 to 20."
NUMBER=INT(20*RND)+1
SEARCH I=1 TO 5
PRINT:PRINT "Your  "S$(I);
INPUT " guess:  ",GUESS
EXITIF GUESS=NUMBER
PRINT "Correct!":PRINT
IF I=1
PRINT "Excellent!"
ELSEIF I=5
PRINT "Good work."
ELSE
PRINT "Very good!"
```

```
ENDIF
ORELSE
IF GUESS > NUMBER
PRINT "Too high."
ELSE
PRINT "Too low."
ENDIF
ENDLOOP
PRINT:PRINT "The number was  " STR$(NUMBER) "."
ENDSRCH
DATA first,second,third,fourth,fifth
```

Did you remember to use Ctrl-Home to save the first 24 lines? If so, save the remaining lines to the edit buffer by using Ctrl-Home. If not, you have to go back and enter the program again.

When you save the second screen, you receive the message **EDIT 20 saved.** The entire sample program is now in the edit buffer.

Let's look at what you entered. You do this by using the LIST command to display the edit buffer on your screen.

Type **L** and press Enter.

The sample program scrolls rapidly up your screen.

Because it is going by too fast for you to read, you use the Space Bar to pause the program. LIST is cancelled by pressing the Enter key.

Press the Space Bar to pause the program. If the file already has scrolled by, repeat the LIST command and press the Space Bar after enough lines appear to fill the screen.

Check these lines to make sure they're correct. When you're done, press any key to continue listing the program.

If you found a mistake, use the EDIT command again to enter full-screen edit mode. If the line with the error is after line 24, specify E and the number of the line with the error.

If you see that you left out lines, use the INSERT command. For example, if you left out line 5, you type **I 4** and press Enter. This tells the program that you want to insert a line after line 4. The line you insert becomes the new line 5.

When you're certain that the sample program is correct, you're ready to save the program.

Saving a File

To save this program, you use the SAVE command. There are several ways to save a file in TFE. You either type the word SAVE or the abbreviation SA, or you can use the F7 and F8 function keys.

If you are saving the program to a diskette in drive A, type **SA DEMO** and press Enter.

If you are saving the program to a diskette in drive B, type **SA B: DEMO** and press Enter.

The default extension for SAVE, .SRC, is added to the filename.

You also can save files by using the function keys. The F7 and F8 function keys supply part of the SAVE command (*SAVE A:* and *SAVE B:*). All you have to supply is the filename.

When you revise a program, you also use SAVE, but you don't have to specify the filename. Because you're using the same filename, the SAVE command copies the new file over the old file.

You now are familiar with some of the commands in the Text File Editor. In the next section, you'll learn about one of the most powerful commands in TFE, the EXECUTE command.

The EXECUTE Command

The EXEC Files

One powerful feature of the Text File Editor and the Structured BASIC Preprocessor is the ability to create and execute files composed of TFE/SBP commands. These are EXEC (execute) files, and they perform multiple editing and preprocessing functions with a single command.

TFE/SBP includes four EXECUTE files, and you may use them as prototypes when creating your own files. The first three — UNPREP, DEBUG, and FREQ — are discussed in Appendix C and Appendix D. The fourth file, AUTOEXEC.TFE, contains all of the commands that are performed at startup time. It contains the 11 commands required to define the 10 function keys and to clear the screen.

With the exception of the EDIT command, which is interactive, you may use any command within an EXECUTE file. You even may execute an EXEC file within an EXEC file if it is the last command in the file.

In addition, all parameters that are valid for commands when entered from the keyboard also are valid for commands you use in an EXEC file.

The AUTOEXEC.TFE File

AUTOEXEC.TFE is a special type of EXEC file. When you load TFE from DOS, AUTOEXEC.TFE automatically executes to define the function keys. You must execute all other EXEC files by entering *EX* and the filename after the command prompt.

Let's look at the AUTOEXEC.TFE file. First, run a directory on the program diskette. You do this by pressing F9.

Notice that all four EXEC files are on the diskette. Now, use the F3 (LOAD A:) key to load the AUTOEXEC.TFE file.

Press F3 (LOAD A:), type **AUTOEXEC.TFE**, and press Enter.

You are prompted **LOAD (Y/N)?** and answer **Y**.

This message appears on your screen:

```
Reading from A:AUTOEXEC.TFE
```

Use the LIST command to look at the file.

Type **L** and press Enter.

The AUTOEXEC.TFE file scrolls up your screen. As the message at the bottom tells you, you can use the Space Bar to pause or the Enter key to cancel. To save you this trouble, we'll show you the entire file:

```
001:KEY 1
002:HELP
003:KEY 2
004:PRINT 0 0 N
```

```
005:KEY 3
006:LOAD A:
007:KEY 4
008:LOAD B:
009:KEY 5
010:GOSUB@
011:KEY 6
012:PRINT
013:KEY 7
014:SAVE A:
015:KEY 8
016:SAVE B:
017:KEY 9
018:FILES A:*.*¦
019:KEY10
020:FILES B:*.*¦
021:STATUS CLS
```

The first 20 lines use the KEY command to set the function keys. The ¦ symbol in lines 2, 18, and 20 makes those keys "hot." These keys don't require that you press Enter to begin execution. For more information about the KEY command, see Chapter 5, "Reference."

The last line uses the STATUS command with the parameter CLS (clear screen) to produce the first screen you see when you start TFE.

Of course, you don't have to use the EXECUTE command every time you want to use the KEY command or the STATUS command. EXEC files simply allow you to execute several commands at the same time.

> **Note:** The AUTOEXEC.TFE file is not a DOS batch file.

Lesson 2. Using SBP

Loading SBP

The Structured BASIC Preprocessor (SBP) and the Text File Editor (TFE) are different components of the same program. Therefore, to use the SBP, you don't have to load a separate program.

If you still have TFE in memory, you're ready to use SBP. If you don't have TFE in memory, type **TFE** after the DOS prompt and press Enter.

What Preprocessing Does to Your Files

A structured BASIC program must be preprocessed so that it can be recognized either by the IBM Personal Computer BASIC Compiler or the BASIC interpreter.

When a structured BASIC program is preprocessed, the following operations are performed.

- The source code is automatically backed-up in a file called BACKUP.TFE before preprocessing begins.
- All macros are expanded.
- Source lines are compressed to gain diskette storage space.
- Remarks are deleted if the remarks are specified by a single quote (').
- All referenced labels are resolved to line numbers.
- The preprocessed code is automatically written to the current filespec unless otherwise specified. The default extension is .BAS.

You'll learn more about preprocessing a file as you progress through the tutorial. Right now, let's look at the structured BASIC DEMO file and indent the structures.

Indenting Structured BASIC Code

Load the DEMO file into memory using the LOAD command in TFE. Type **LO B:DEMO** and press Enter.

Now, list this program. Type **L** and press Enter.

This file is in structured BASIC. Notice the macros and the lack of line numbers. To further show you the structures, let's use a command in the SBP, the INDENT command. This command performs automatic indentation on structured code.

The INDENT command begins indentation in column 1 on the first line of a program and returns to column 1 at the end of a program. If the final line in your file does not begin in column 1, you know that a macro is probably missing or out of sequence.

This command indents each line of code within a structure the specified number of spaces.

The INDENT command has three parameters: the first line you're indenting, the last line you're indenting, and the amount, or number of columns, you want the structures indented. The default values for the line parameters are the first and last lines of a file. The default value for *amount* is two columns.

Let's try the INDENT command on the DEMO file. Use the default values on this program.

Type **IND** and press Enter.

This indents all the structures two spaces.

When the INDENT command finishes, use the LIST command to look at your program. Type **L** and press Enter.

The file now looks like this:

```
B:DEMO.SRC — 01:21:38 — 01-01-1980

KEY OFF:CLS'Number Guessing Game
RANDOMIZE VAL (RIGHT$TIME$,2))
I=0:DIM S$(5)
DO
  I=I+1
LEAVE I>5
  READ S$(I)
ENDDO I=0
PRINT "I've got a number from 1 through 20."
NUMBER=INT(20*RND)+1
SEARCH I=1 TO 5
  PRINT:PRINT "Your  "S$(I):
  INPUT " guess:  ",GUESS
EXITIF GUESS=NUMBER
  PRINT "Correct!":PRINT
  IF I=1
    PRINT "Excellent!"
  ELSEIF I=5
    PRINT "Good work."
  ELSE
    PRINT "Very good!"
  ENDIF
ORELSE
  IF GUESS>NUMBER
    PRINT "Too high."
  ELSE
    PRINT "Too low."
  ENDIF
ENDLOOP
  PRINT:PRINT "The number was "STR$(NUMBER)"."
ENDSRCH
DATA first,second,third,fourth,fifth
```

The INDENT command has made the program's structure readily apparent. By studying the placement of the macros, you can follow the flow of control in the program.

Now that you have seen what a structured BASIC program looks like, you're ready to examine preprocessing.

Preprocessing a File

You use the PREP command to preprocess a file. This command has two parameters. Its syntax looks like this:

PREP [* /*filespec*(.BAS)][* /*Nobackup*]

The first optional parameter specifies the output file, and the second optional parameter specifies the backup file.

In the first parameter, the * sign saves the preprocessed file to the current filename and uses .BAS as the extension. If you decide not to use the current filespec, you can specify a new one. When you specify a new filespec, you either can specify a different extension or use the default extension, .BAS.

In the second parameter, the * sign backs-up your source code using the current filespec (including the current extension). If you don't want a backup, you can use the nobackup option. You specify nobackup with the letter **N**.

If you don't specify any of the options in PREP, the command saves your preprocessed code to the current filename with .BAS as the extension. It also creates a backup file of your source called BACKUP.TFE. This file is written to the drive associated with the current filespec.

An important point to remember about the PREP command is that it changes the contents of the edit buffer. You are asked **PREP (Y/N)?** to be sure you have saved your source and are ready to run SBP. This is to prevent you from losing the contents of the edit buffer.

For the DEMO.SRC example, let's specify * * as the parameters.

Type **PREP** * * and press Enter.

The question, **PREP (Y/N)?**, appears on the screen. Because you've saved your file on DEMO.SRC, you reply yes.

Type **Y** and press Enter.

You now see the preprocessor expand the macros, compress the code, remove the remarks, and resolve the label references.

Finally, the command saves the results of the preprocessing to the current filespec and adds .BAS as the extension. You now have a new file, DEMO.BAS.

When the preprocessing is done, list the edit buffer. What you see is a mangled form of your structured program in executable BASIC form. This form has no labels, no macros, no remarks, and is not very readable. It, however, is in a form acceptable to BASIC.

To run this program, you have to end TFE by using the QUIT command. Type **Q** and press Enter. You are asked **QUIT (Y/N)?** to insure that you want to erase the current edit buffer and end the program. Since PREP saved the preprocessed DEMO file to DEMO.BAS, type **Y** and press Enter.

When you return to DOS, type **BASIC B:DEMO** and press Enter. The screen clears and then displays this:

```
I've got a number from 1 through 20.

Your first guess:
```

You should play the game to verify that the functions and the messages actually work.

Besides the PREP command, there are two other ways to preprocess a file. These methods have specific uses and are discussed in Appendix D, "Other Ways to Preprocess."

You're now ready to move on to the next Lesson. In Lesson 3, you'll learn how to use the BASIC Formatter and Cross-Reference utilities.

Lesson 3. Using FRMTBAS

In this lesson, you'll learn how to use the BASIC Formatter and Cross-Reference (FRMTBAS) program to produce formatted and cross-referenced listings of your program.

Remember, you cannot use FRMTBAS on structured BASIC programs. FRMTBAS only works on unstructured BASIC programs that have an ASCII format or on preprocessed structured BASIC programs.

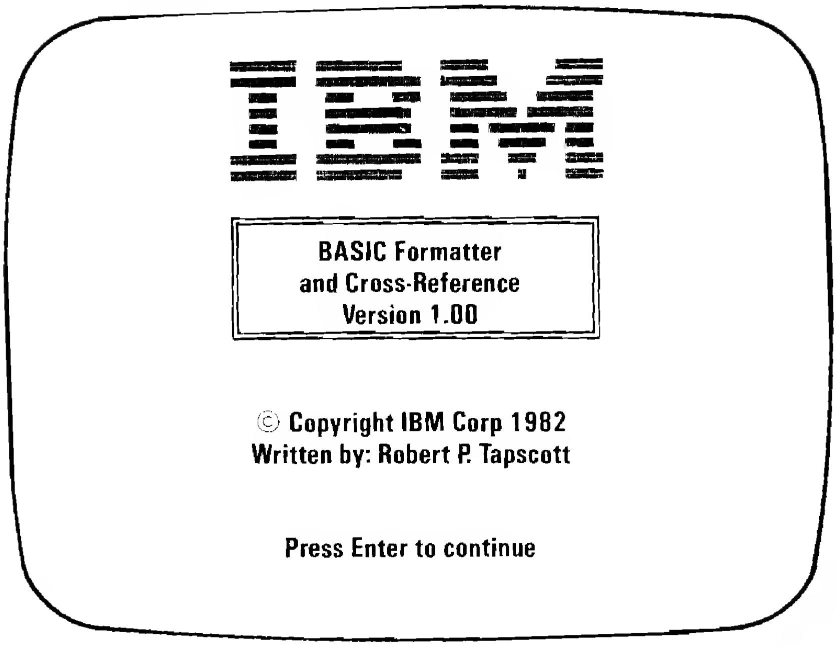
Loading FRMTBAS

To load FRMTBAS, insert the diskette with FRMTBAS on it in drive A and turn on the computer. This diskette must have DOS on it. If it doesn't, see Chapter 2 for directions on adding DOS to your diskette.

If you are in TFE, return to DOS by using the QUIT command. Type **Q** and press Enter. When asked, **QUIT (Y/N)?**, answer **Y**.

When the DOS prompt (**A >**) appears on the screen, type **FRMTBAS** and press Enter.

The following coypright message appears on the screen:

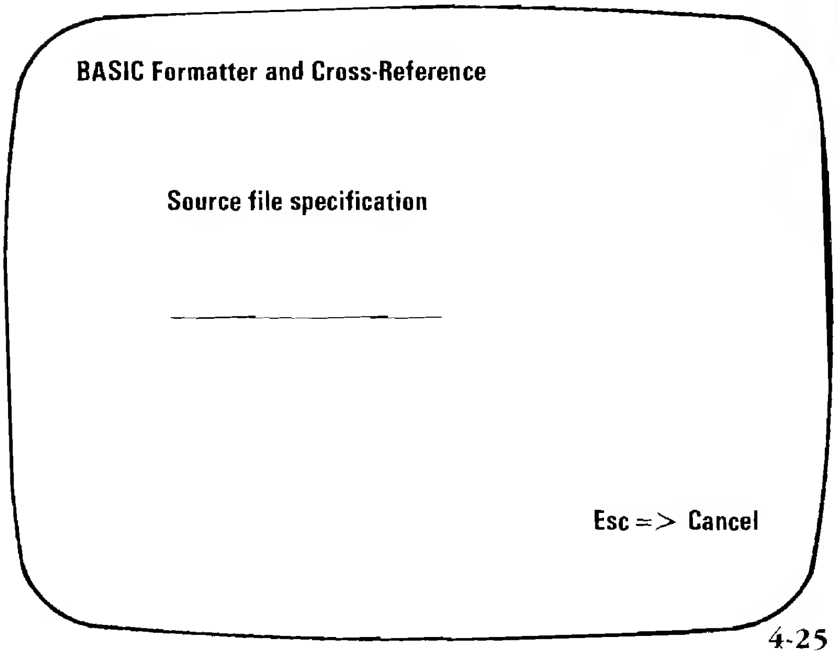

BASIC Formatter
and Cross-Reference
Version 1.00

© Copyright IBM Corp 1982
Written by: Robert P. Tapscott

Press Enter to continue

This copyright notice remains on the screen until you press Enter. Do so now.

The first screen appears and looks like this:


BASIC Formatter and Cross-Reference

Source file specification

Esc => Cancel

The words **Esc => Cancel** appear at the bottom of the screen. These instructions appear on every screen in FRMTBAS. If you want to cancel FRMTBAS, press the Esc (Escape) key. You then return to DOS.

Specifying a File

This first screen asks you for the name of the file you wish to cross-reference and/or format. You need to specify the device name (unless your file is on a diskette in the default drive, in our case, drive A), the filename, and the extension. In FRMTBAS, the default extension is .BAS.

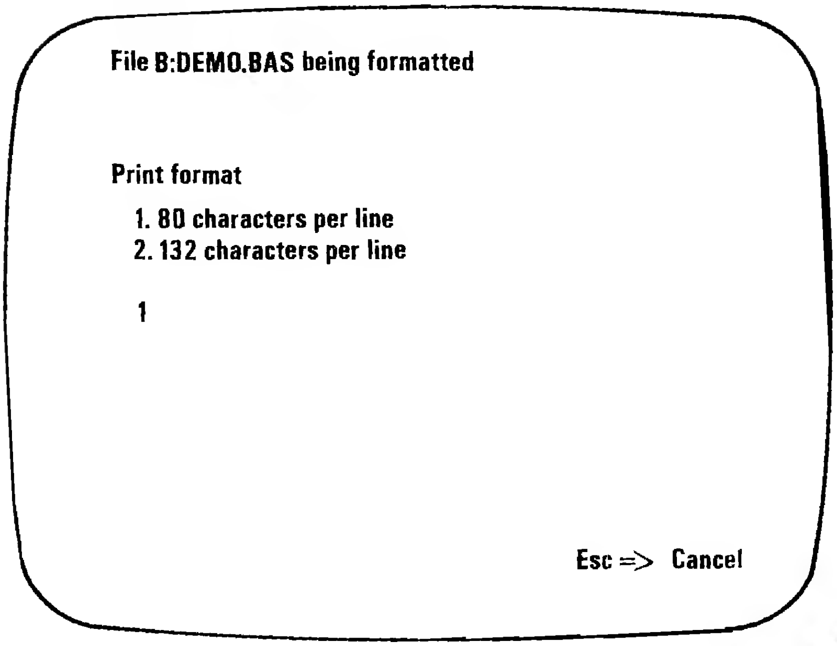
You are going to specify the preprocessed DEMO.BAS file. Because you created this file on TFE, it already is in ASCII format. Before you specify a file, always be sure it is in ASCII format. If you need to convert a file to ASCII format, follow the directions included in Appendix B, "Converting Files to ASCII Format."

To specify the DEMO file, you have to give the drive name (B:) and the filename (DEMO). You don't have to include the extension because it defaults to .BAS.

Type **B:DEMO** and press Enter.

If you ever make a mistake typing something in FRMTBAS, you don't have to cancel the program if you haven't pressed Enter yet. You can use the Backspace key to delete the error and then type the correct entry. If you have already pressed Enter, you have to cancel the program and begin again.

Once you've typed and entered the correct filespec, the file is loaded into memory. Your next screen looks like this:



```
File B:DEMO.BAS being formatted


Print format
  1. 80 characters per line
  2. 132 characters per line

  1





                                  Esc => Cancel
```

Notice that in the upper lefthand corner of this screen a message appears telling you that **File B:DEMO.BAS being formatted**. All of the menu screens contain this message, even though your program is not being formatted at the present time. Instead, you're choosing a format by specifying the print format, the output, and the cross-references you want.

On this second screen, you choose the print format you want.

Choosing a Print Format

You have a choice between two print formats. Option 1 is 80 characters per line (actually 79 to avoid a blank line). Option 1 causes in-line remarks to appear starting in column 35. Option 2 is 132 characters per line (actually 131) and is useful for a program with very long statements or character strings. With option 2, in-line comments appear starting in column 65.

To get option 1, you either type the number 1 or simply press the Enter key, because option 1 is the default value. To get option 2, you must type the number 2 and then press the Enter key.

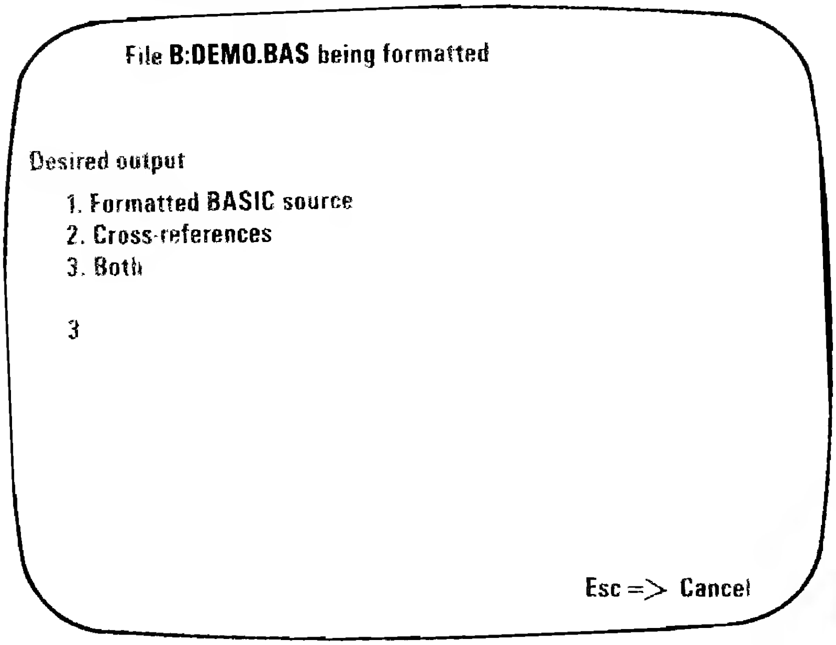
If you have statements, comments, or character strings that are too long to fit either format, the utility divides them. Statements are divided at the nearest control character before the end of the line. Character strings are divided at the end of the line. Comments are divided after a blank, a comma, or a semicolon if one is within the last 10 characters before the end of the line; otherwise, comments are divided at the end of the line.

The BASIC Formatter shows divided lines by placing the second part on the following line and indenting it three positions to the right of the preceding output line. This style enables you to see at a glance that a line has been divided.

Select option 1 for the DEMO program. To do this, press the Enter key.

Determining the Desired Output

After choosing the print format, you are asked to decide on the type of output you want. The screen looks like this:



```
        File B:DEMO.BAS being formatted


Desired output
   1. Formatted BASIC source
   2. Cross-references
   3. Both

   3








                                                  Esc => Cancel
```

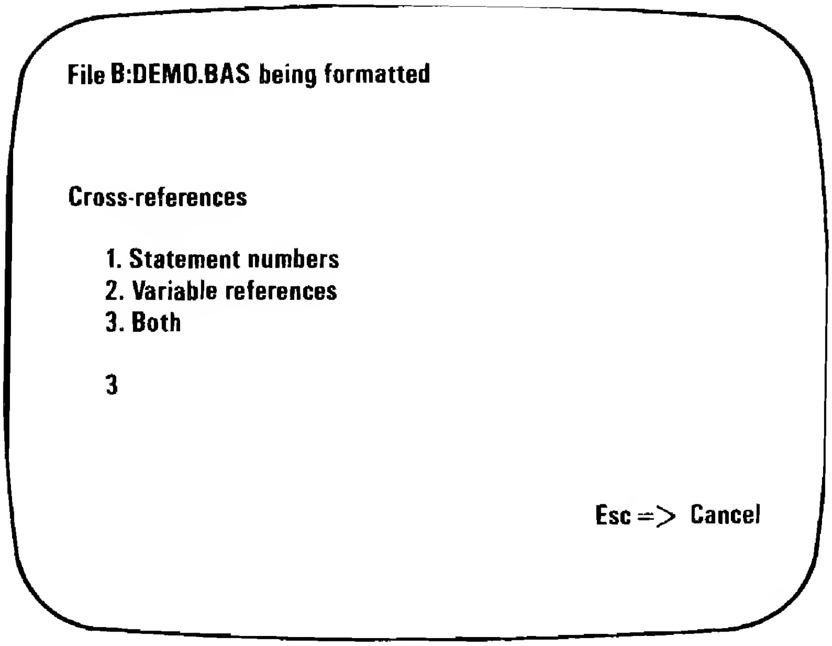
In this case, choose the default value, option 3. By selecting option 3, you receive both the formatted BASIC source program and the cross-reference listings. If you choose option 1, you receive only the formatted source code listing. Likewise, if you choose option 2, you receive only the cross-reference listings.

Type **3** and press the Enter key.

Selecting Cross-Reference Listings

Because you chose option 3 on the previous screen, you now must pick the type of cross-reference listings you want. You also are given this choice if you chose option 2 on the previous screen.

At the moment, your screen should look like this



```
File B:DEMO.BAS being formatted



Cross-references

    1. Statement numbers
    2. Variable references
    3. Both

    3



                                             Esc => Cancel
```

You can obtain a cross-reference listing of the statement numbers, the variable references, or both.

Option 3 is the default choice on this screen and produces both a listing of the statement numbers and the variable references. If you choose option 3, you only have to press Enter. If you choose either of the other two options, you have to type the number and then press Enter.

So that you can see both listings, choose option 3 to run on the DEMO program. Since this is the default value, just press the Enter key.

If you select option 2 or option 3, you have to pick the reserved words you want referenced. Since you chose option 3, you have to indicate the reserved words you want referenced.

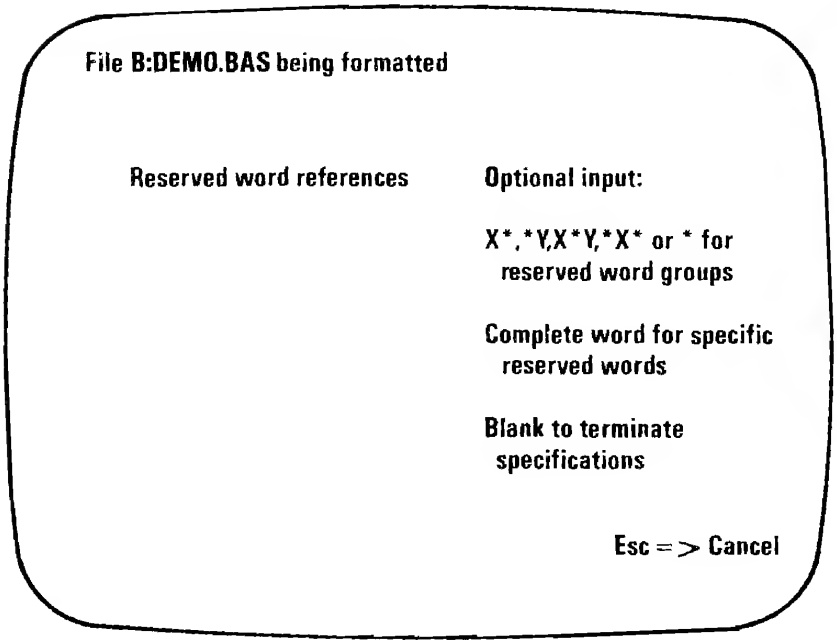
Selecting Reserved Words

Table Size Limits

When you pick reserved words, the tables have size limits, so keep the following figures in mind as you select the number of reserved words you want.

- You can have 1024 entries in the statement number cross-reference table.
- You can have 256 character strings, each with up to 127 variable names depending on length, in the variable name table.
- You can have 3072 entries in the variable cross-reference table.

The reserved words screen looks like this:



```
File B:DEMO.BAS being formatted


     Reserved word references        Optional input:

                                     X*,*Y,X*Y,*X* or * for
                                       reserved word groups

                                     Complete word for specific
                                       reserved words

                                     Blank to terminate
                                       specifications

                                                  Esc => Cancel
```

In response to the prompt, you may enter a specific reserved word or variations for groups of reserved words.

Reserved Word Syntax

Let's examine the different types of variations you can enter.

X* Use when you want a specific prefix and any suffix. The * symbol means that any characters can occupy that position.

Example: PR* finds all reserved words beginning with the letters PR.

***Y** Use when you want a specific suffix but do not care what precedes it.

Example: *PR finds reserved words that end with the letters PR.

X*Y Use when you are looking for specific starting and/or ending characters but do not care what characters are in the middle.

Example: P*R finds reserved words beginning with P and ending with R.

X Use when you are looking for a specific character or group of characters in the middle of a word. With this form, anything can precede and follow the character(s) you specify.

Example: *PR* finds reserved words with the letters PR in the middle of them.

* Use to find all of the reserved words.

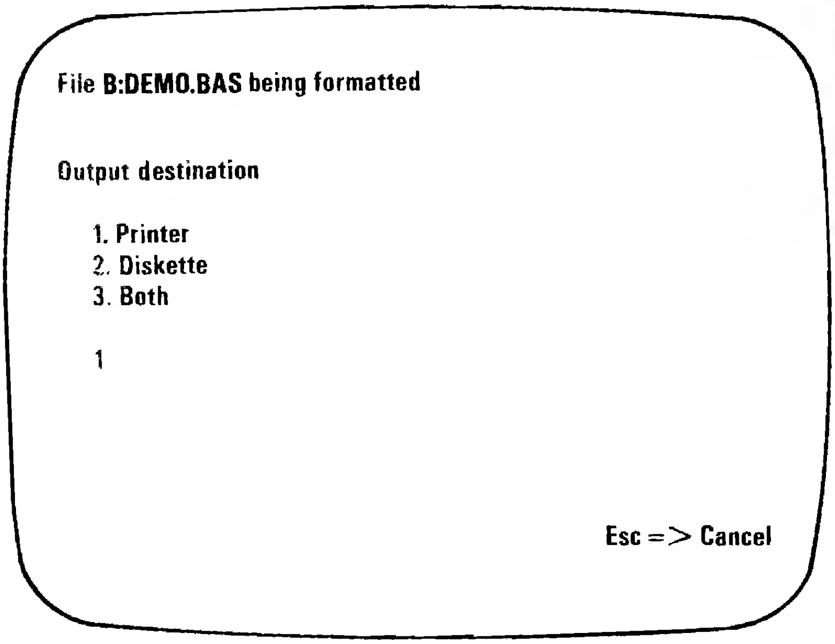
Note: Those reserved words such as PRINT# that may be written as either PRINT# 1 or PRINT #1 are handled by ignoring the # symbol. Thus INPUT#, PRINT#, and WRITE# are treated as PRINT, INPUT, and WRITE, respectively.

On the DEMO program, use the reserved words FOR and NEXT. Type **FOR** and press Enter. Then type **NEXT** and press Enter. Now, press Enter without specifying anything. This signifies that you are finished. You move on to the next screen.

Specifying the Output Destination

Regardless of whether you choose to have a cross-reference listing or not, the utility now asks you for the output destination.

The screen looks like this:



```
File B:DEMO.BAS being formatted


Output destination

   1. Printer
   2. Diskette
   3. Both

   1




                                        Esc => Cancel
```

As the screen shows, your choices are the printer (hard copy only), the diskette (storage file copy only), or both. The default choice in this case is option 1, the printer.

Choose the printer as an output destination if you want immediate output and a hard-copy of your material.

Choose the diskette as one of the options if you want to delay output.

Another advantage to choosing the diskette as one of the output destinations (either with option 2 or option 3) is that the file you produce can be used as input to the BASIC Compiler or interpreter.

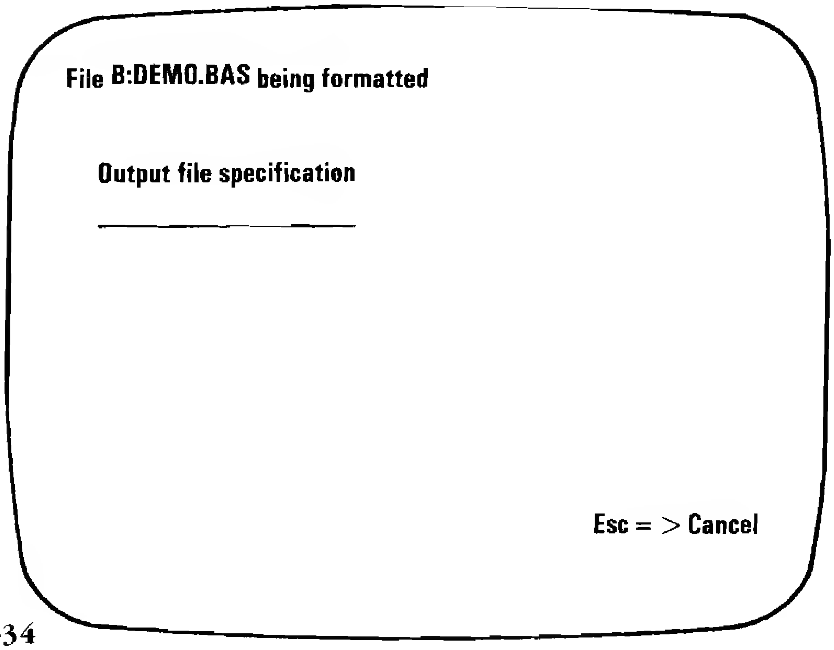
To see how both options work, pick option 3 to use on the DEMO program.

Type 3 and press Enter.

Giving a File Specification

Since you chose to have diskette output as well as printer output, you have to give the diskette output a filename.

Your screen looks like this:



```
File B:DEMO.BAS being formatted

   Output file specification

   ______________________





                                        Esc = > Cancel
```

You need to supply the drive letter that your source diskette is in and then the filename. The form used is given below:

d:filename. ext

The *d:* stands for the letter of the drive that your source program is in (where the file is going to be sent). If, for example, you want the file to be placed on a diskette in drive B, you type *B:* and then the filename. The default extension is .FTD, signifying that the file has been formatted.

Your output diskette file specification, however, cannot be the same as the source file. You must have a different filename or extension.

Another important fact to remember is that because of the operations you choose to have done on the source file, your output file may be larger (three to five times larger) than your source file. You have to allow enough diskette space to hold this larger file.

Therefore, if your source file is fairly large, direct your output file to a relatively blank diskette to allow room for the expanded file.

Be certain you have enough room on your diskette for the DEMO file. Then type **B:DEMO** and press Enter.

The program adds the default extension .FTD to this file.

You now are through designating the options you want to use, and the processing of your source file begins.

Processing

To inform you of the utility's progress, this message appears on the screen:

```
Current line –
```

At this point, the utility is formatting and searching the lines for references. Printing (both to the printer and to the diskette) occurs here because you chose the formatting option.

The number of the line currently being processed appears after the words **current line.** You can tell how compact your source code is by watching how fast the source code line numbers change.

Since you chose the cross-reference option for statement numbers (option 3), this message appears next:

```
Sorting the statement table
```

The statement numbers are being sorted in numerical order.

The sorting of the statement number table is needed for efficient output. There is a delay while the statement numbers are sorted.

When the utility has finished sorting the statement numbers, the following message appears with changing line numbers:

```
Working on statement –
```

The statement references are printed now. The utility lets you know which statement is being worked on at the moment.

The next screen appears because you chose to have variable references produced. The variable references are printed now.

You get this message:

```
Working on variable –
```

When the variable references have finished printing, FRMTBAS is through processing your program.

The following message appears on the screen:

```
FRMTBAS completed. Press Enter to continue.
```

When you press Enter, you either return to DOS (if it is on your program diskette), or you are instructed to insert the DOS diskette.

You now have a new file on your diskette, DEMO.FTD. You also received a printout of the formatted code and the references. The printouts you should have received are shown on the next pages. Compare them with the ones you got.

```
DEMO                               03:00:28   01-01-1980                    Page 1
      IBM Personal Computer BASIC Formatter and Cross-Reference  V 1.00

   1  KEY OFF:
      CLS:
      RANDOMIZE VAL (RIGHT$(TIME$,2)):
      I=0:
      DIM
         S$(5):
      QO=0:
      WHILE QO=0:
      ¦  I=I+1:
      ¦  IF
      ¦     I>5
      ¦        THEN
      ¦           ( GOTO ) 2
      ¦        ELSE
      ¦           READ S$(I):
      ¦           QO=I=0:
      WEND
```

```
2  PRINT"I've got a number from 1 through 20.":
   NUMBER=INT(20*RND)+1:
   FOR I=1 TO 5:
   ¦  PRINT:
   ¦  PRINT"Your "S$(I);:
   ¦  INPUT" guess: ",GUESS:
   ¦  IF
   ¦     (GUESS=NUMBER)=0
   ¦        THEN
   ¦           ( GOTO ) 5
   ¦        ELSE
   ¦           PRINT"Correct!":
   ¦           PRINT:
   ¦           IF
   ¦              I=1
   ¦                 THEN
   ¦                    PRINT"Excellent!":
   ¦                    GOTO 4
3  ¦  IF
   ¦     I=5
   ¦        THEN
   ¦           PRINT"Good work."
   ¦        ELSE
   ¦           PRINT"Very good!"
4  ¦  GOTO 7
```

```
5  ¦  IF
   ¦     GUESS>NUMBER
   ¦        THEN
   ¦           PRINT"Too high."
   ¦        ELSE
   ¦           PRINT"Too low."
6  NEXT I:
   PRINT:
   PRINT"The number was "STR$(NUMBER)"."
7  DATA
      first,
      second,
      third,
      fourth,
      fifth
```

```
DEMO                          03:00:28   01-01-1980                        Page 3
      IBM Personal Computer BASIC Formatter and Cross-Reference  V 1.00

 Summary
Input line count     -     7
Statement count      -    38
Line references      -     4
Variable references  -    26
```

```
DEMO                          03:00:31   01-01-1980                        Page 4
      IBM Personal Computer BASIC Formatter and Cross-Reference  V 1.00
Statement References

        2         1
        4         2
        5         2
        7         4
```

```
DEMO                          03:00:31   01-01-1980                 Page 5
      IBM Personal Computer BASIC Formatter and Cross-Reference  V 1.00

Variable   References

FOR             2

GUESS           2        2        5

I               1        1        1        1        1        1        2
                2        2        3        6

NEXT            6

NUMBER          2        2        5        6

QO              1        1        1

S$              1        1        2
```

Chapter 5. Reference

Contents

Using the Reference Section

This chapter describes each of the Text File Editor commands. The commands are listed in alphabetical order to give you quick access to the command information you need.

The reference material is arranged in the following order:

Purpose:	An explanation of what the command does.
Format:	A listing of the syntax of the command, including the parameters.
Remarks:	Information on uses, special features, and qualities of the command. This includes warnings and reminders.
Example:	An example of how to use the command.

Syntax Description

The syntax diagrams for the commands have specific conventions:

- Words or letters in uppercase are keywords, and you must enter them as shown. You may enter them in either uppercase or lowercase.
- You supply any items shown in lowercase italics. This includes filenames, parameters, etc.
- Square brackets, [ ], identify optional parameters that may not begin with 0 through 9 or contain blanks.
- Braces, { }, identify parameters that are optional at that place. You are prompted for this parameter later if you do not enter it initially.
- Parentheses, (), identify default file extensions.

Commands

The following is a list of all the TFE/SBP commands and their parameters. The command abbreviations are in uppercase.

Add	*{string} first last type*
CEnter	*first last width*
Change	*{old} {new} first last type*
COMbine	*first last separator*
COpy	*first last after*
Delete	*first last*
Edit	*firstline firstcol maxlen*
EXecute	*filespec*
EXPand	*line separator*
FILes	*filespec*
Find	*{string} first last type*
FLow	*first last width*
Help	*commandname*
INDent	*first last amount*
Insert	*after*
Key	*{string} keynumber*
KIll	*filespec*
List	*first last width/Nonumber*
LOad	*filespec* (.SRC) [*Append*] *first last*
Move	*first last after*
NAme	*oldfilespec* AS *newfilespec*
New	
PREp	[*/*filespec* (.BAS)] [*/*Nobackup*]
Print	[*filespec* [*Notitle*]] *first last width/Nonumber*
Quit	
Repeat	*first last times*
SAve	[*d:/filespec* (.SRC) [*Append*]] *first last*
SHift	*first last amount*
Status	[*CLS*]
Type	*filespec* (.SRC) *first last*

Reference

ADD Command

Purpose: Allows a string to be added to each line in a range of lines.

Format: A {*string*} *first last type*
or
ADD {*string*} *first last type*

Remarks:

string	is a series of characters you wish to add. If you don't supply the string originally, you are prompted for it.

In the case of number and sequence types, the string is the starting number and the optional increment amount.

first, last	are line numbers in the range of 1 to 999. *first* is the first line to which you're adding a string. *last* is the last line to which you're adding a string.
type	is the type or kind of ADD operation you want to make.

The type parameter determines where the string is added.

The four ADD types are as follows:

- *Beginning of line* — adds a string to the beginning of the lines. You specify it with the letter **B.**

- *End of line* (default) — adds a string to the end of the lines. You specify it with the letter **E.**

- *Number at beginning of line* — adds numbers to the beginning of the lines and increments by any amount you specify. You specify it with the letter **N.**

- *Sequence number at end of line* — adds numbers to the end of the lines and increments by any amount you specify. You specify it with the letter **S.**

Example: `A 1 15 B`

Adds a string to the beginning of lines 1 through 15. The following prompt appears:

```
String:
```

You now enter the string you want added to lines 1 through 15.

CENTER Command

Purpose: Centers one or more lines of the edit buffer.

Format: CE *first last width*

or

CENTER *first last width*

Remarks:

first, last	are line numbers in the range of 1 to 999. *first* is the first line you're centering. *last* is the last line you're centering.
width	is the width of the line in which the data is centered. If no width is given, the screen width minus 1 is used.

Example:

You are asked to verify this command.

If you answer **Y**, lines 1 through 10 are centered within 80-character lines.

CHANGE Command

Purpose: Allows you to replace a specific character string with another string.

Format: C {*old*} {*new*} *first last type*

or

CHANGE {*old*} {*new*} *first last type*

Remarks

old	is the string you're replacing. If you don't supply the string originally, you are prompted for it.
new	is the new string. If you don't supply the string originally, you are prompted for it.
first, last	are line numbers in the range of 1 to 999. *first* is the first line you want searched for changes. *last* is the last line you want searched for changes.
type	is the type or kind of search operation you are performing.

The type parameter restricts or narrows the way in which the CHANGE command searches a file. This saves you time when you are searching for a specific type of character string.

You must enter both the old and the new strings exactly as they appear or will appear. This means that if the characters are in uppercase, you must enter them in uppercase.

There are 12 different types of CHANGE operations to choose from. Each one searches for character strings in a specific manner.

Here are the 12 CHANGE types:

- *Characters* (default) — searches all characters for the specified string. You specify it with the letter **C**.

- *Prefix of word* — searches for strings only at the beginning of words. You specify it with the letter **P**.

- *Suffix of word* — searches for strings only at the end of words. You specify it with the letter **S**.

- *Word* — searches for strings that are complete words. You specify it with the letter **W**.

- *Token* — searches for complete words that are outside double quotes. It finds reserved words, variables, constants or any other acceptable symbols. You specify it with the letter **T**.

- *Numeric following* — searches for strings that are followed by a numeric character, 0 through 9. You specify it with the letter **N**.

- *Beginning of line* — searches for the string only at the beginning of lines (column 1). It cannot be used on indented lines. You specify it with the letter **B**.

- *End of line* — searches for the string only at the end of lines. You specify it with the letter **E**.

- *First non-blank* — searches for the string by looking at the first non-blank characters in each line. It searches beyond blanks and, therefore, can be used on indented lines. You specify it with the letter **F**.

- *Starting column* — searches for strings that begin in a specified column. This is like the beginning of line type, but starting column allows you to search in any column. You specify it with the column number.

- *Inside quotes* — searches for strings inside double quotes only. You specify it with the letter **I**.

- *Outside quotes* — searches for strings outside double quotes only. You specify it with the letter **O**.

Example: `C 1-20`

You are prompted for the old string:

```
OLD REM
```

(String must be entered exactly as it appears.)

You now supply the new string:

```
NEW '
```

You are using the default type, *character*. This searches all characters in lines 1 through 20 and replaces any occurrence of REM with a single quote (').

COMBINE Command

Purpose: Joins a group of lines into one line with a maximum of 230 characters.

Format: COM *first last separator*
or
COMBINE *first last separator*

Remarks:

first, last	are line numbers in the range of 1 to 999. *first* is the first line you're combining. *last* is the last line you're combining. If the *first* and *last* are not specified, they default to the first and last lines of the program.
separator	is the non-blank character or characters you specify to delimit (separate) the statements. The default is a colon.

Important: The COMBINE command does not look at the lines it combines and may change the flow of control.

Example: [illegible]

Combines lines 1 through 3 into a single line, separating the statements with colons.

COPY Command

Purpose: Allows a single line or a group of lines to be duplicated and inserted after any existing line in the file. The editing line numbers then are renumbered. This command allows you to place the same lines in several locations.

Format: CO *first last after*

or

COPY *first last after*

Remarks:

first, last	are line numbers in the range of 1 to 999. *first* is the first line you're copying. *last* is the last line you're copying.

To copy only one line, you specify it as both the first and the last line.

after	is the line after which the specified lines are copied.

To copy lines to the beginning of the file, you specify that they be copied after line 0.

Note: You must enter all three parameters.

Example: `CO 1 5 10`

Copies lines 1 through 5 and places them after line 10. These lines now become the new lines 11 through 15 in the edit buffer. The old lines from 11 on are renumbered.

DELETE Command

Purpose: Removes one or more lines from the edit buffer and renumbers the remaining lines.

Format: D *first last*
or
DELETE *first last*

Remarks: *first, last* are line numbers in the range of 1 to 999. *first* is the first line you're deleting. *last* is the last line you're deleting.

Note: DELETE will not erase the entire file. Also, if you do not supply a last line parameter, only the line you specify as the first parameter is deleted.

Example:

Deletes lines 10 through 12 in the edit buffer and renumbers the remaining lines.

EDIT Command

Purpose: Changes the line editor to a full-screen editor.

Format: E *firsline firstcol maxlen*
or
EDIT *firstline firstcol maxlen*

Remarks:

firstline	is a line in the range of 1 to 999. *firstline* is the first line you're editing. The default is line 1.
firstcol	is the column that initially appears in the upper lefthand corner. The default is column 1.
maxlen	is the maximum line length. The default is the last maxlen specified, or 255 characters per line when the full-screen editor is first started.

The EDIT command truncates lines if any line in a file is longer than the specified maxlen. You're asked, **Truncate lines (Y/N)?** before the file is loaded. If you specify **Y**, all lines are truncated at the specified maxlen and you enter edit mode. If you specify **N**, the EDIT command is cancelled.

The space required for full-screen editing is 24 * (maxlen + 3) bytes and is not included in the normal edit buffer. If you have limited space and are editing lines less than 255 bytes, you can acquire more space by using EDIT and specifying a smaller maxlen.

With the EDIT command, the following keys have new definitions:

Ctrl-Home	Saves the results of the full-screen editing into the standard edit buffer and exits full-screen mode. This gives you the message **EDIT n saved** when you exit the edit mode.
Ctrl-PgDn	Cancels the full-screen editing without saving the changes. This gives you the message **EDIT n cancelled** when you exit the edit mode.
Ctrl-→	Scrolls the screen right 1/2 the width of the screen, if possible.
Ctrl-←	Scrolls the screen left 1/2 the width of the screen, if possible.

In addition, the cursor wraps around the screen, top to bottom and side to side.

The rest of the keys have the same definitions they have in BASIC.

Example:

```
E
```

Lets you edit, beginning with line 1, column 1, with a line length of 255.

```
E 4 3 80
```

Let's you edit, beginning with line 4, column 3, with a line length of 80.

EXECUTE Command

Purpose: Allows you to execute EXEC files that can perform multiple functions with a single command.

Format: EX *filespec*

or

EXECUTE *filespec*

Remarks:

filespec	is the file specification of the EXEC file you want to use.
	filespec defaults to the DOS drive (usually drive A).

There are four EXEC files in TFE/SBP: EX UNPREP "unpreprocesses" files; *EX DEBUG* preprocesses a file for debugging (error correction); *EX FREQ* preprocesses a file for frequency testing; and *AUTO-EXEC. TFE,* a special type of EXEC file, defines the function keys and clears the screen when you start TFE.

Example:

```
EX DEBUG
```

Executes the commands required to preprocess a program in DEBUG mode.

EXPAND Command

Purpose: Expands, or divides, a multi-statement line into several single statement lines.

Format: EXP *line separator*
or
EXPAND *line separator*

Remarks:

line	is the line you're expanding.
separator	is the non-blank character or characters you're using to divide the statements. The default is a colon.

Important: The EXPAND command does not check for invalid expansions. For example, IF statements are expanded without regard for the flow of control.

Example:
```
EXP 1
```
Expands line 1, placing each statement on an individual line wherever a colon appears.

FILES Command

Purpose: Displays the name of some or all of the files on a diskette.

Format: FIL *filespec*

or

FILES *filespec*

Remarks: *filespec* is the file specification of the file you want displayed. *filespec* defaults to the DOS drive (usually drive A).

If you don't give a device name or filename, all files on the diskette in the default drive are displayed.

At startup, F9 and F10 are set to display all files on the diskettes in drive A and drive B, respectively.

If you want a listing of the files in the default drive, just use the FILES command without parameters. If you want to display the files in another drive, drive B for example, type **FIL B:*.*** and press Enter. The *.* signifies that you want a listing of all the files on the specified diskette (in this case, the diskette in drive B).

You can restrict the filenames that are displayed by the FILES command. Refer to the FILES command in your IBM Personal Computer *BASIC* manual for further information.

Example: `FIL B:*.*`

Displays all files on the diskette in drive B.

```
FIL
```

Displays all files on the diskette in the default drive, usually drive A.

FIND Command

Purpose: Allows you to find specific character strings quickly.

Format: F {*string*} *first last type*
or
FIND {*string*} *first last type*

Remarks:

string	is a series of characters you wish to find. If you don't supply the string originally, you are prompted for it.
first, last	are line numbers in the range of 1 to 999. *first* is the first line in which you wish to find a string. *last* is the last line in which you wish to find a string.
type	is the type or kind of FIND operation you are performing.

The type parameter restricts or narrows the way in which the FIND command searches a file. This saves you time when you are searching for a specific type of character string.

You must enter the string exactly as it appears in the file. This means that if the characters are in uppercase, you must enter them in uppercase.

There are 12 different types of FIND operations to choose from. Each one searches for character strings in a specific manner.

Here are the 12 FIND types:

- *Characters* (default) — searches all characters for the specified string. You specify it with the letter **C**.
- *Prefix of word* — searches for strings only at the beginning of words. You specify it with the letter **P**.
- *Suffix of word* — searches for strings only at the end of words. You specify it with the letter **S**.
- *Word* — searches for strings that are complete words. You specify it with the letter **W**.
- *Token* — searches for complete words (blanks on both sides) that are outside double quotes. It finds reserved words, variables, constants, or any other acceptable symbols. You specify it with the letter **T**.
- *Numeric following* — searches for strings that are followed by a numeric character, 0 through 9. You specify it with the letter **N**.

- *Beginning of line* — searches for the string only at the beginning of lines (column 1). It cannot be used on indented lines. You specify it with the letter **B**.

- *End of line* — searches for the string only at the end of lines. You specify it with the letter **E**.

- *First non-blank* — searches for the string by looking at the first non-blank characters in each line. It searches beyond blanks and, therefore, can be used on indented lines. You specify it with the letter **F**.

- *Starting column* — searches for strings that begin in a specified column. Starting column allows you to search in any column. You specify it with the column number.

- *Inside quotes* — searches for strings inside double quotes only. You specify it with the letter **I**.

- *Outside quotes* — searches for strings outside double quotes only. You specify it with the letter **O**.

Example:
```
F 1 30 B
```

You are prompted for a string.

```
String:REM
```

(String must be entered exactly as it appears.)

This searches for the character string REM at the beginning of lines 1 through 30.

FLOW Command

Purpose: Combines or "flows together" one or more lines or an entire file into paragraphs.

Format: FL *first last width*

or

FLOW *first last width*

Remarks:

first, last	are line numbers in the range of 1 to 999. *first* is the first line in the group to be flowed. *last* is the last line in the group to be flowed.
width	is the width of the paragraphs. The default is the width of the screen minus one.

To show where a paragraph begins, you place a blank space in column 1 (the first space on the left side of the screen). You show paragraph indentation by putting blank spaces before the first word of the paragraph.

This command does not remove blank lines, but it does remove blank spaces within lines.

Do not use the FLOW command on any line that contains a figure or a diagram.

Example:

```
FL 1 14 80
```

Flows lines 1 through 14 into 80-column wide paragraphs.

HELP Command

Purpose: Displays information about the commands.

Format: H or H *commandname*
or
HELP or HELP *commandname*

Remarks: *commandname* is the name or the acceptable abbreviation for a command.

If you type **H** or **HELP**, you get a description of the HELP command. You use the HELP command to get specific information about a command by typing **H** or **HELP** and the name of a command or the abbreviation for a command.

Example:

Gives you the syntax for the structured macros.

This appears on your screen:

DO Structure	DO [condition/@variable] LEAVE condition ENDO [condition]
SEARCH Structure	SEARCH [condition/@variable] EXITIF condition ORELSE ENDLOOP [condition] ENDSRCH

INDENT Command

Purpose: Indents the macros in structured BASIC source.

Format: IND *first last amount*
or
INDENT *first last amount*

Remarks:

first, last	are line numbers in the range of 1 to 999. *first* is the first line you're indenting. *last* is the last line you're indenting. If *first* and *last* are not specified, they default to the first and last lines of the program.
amount	specifies the number of columns to indent when a macro is encountered in the specified range of lines.

If no macro is encountered, then no indentation occurs. *first* and *last* must be specified if you want to specify an amount. The default for amount is two. IND 0 0 0 can be used to strip all blanks from the beginning of every line in the file.

Example:

```
IND
```

Indents, in increasing increments of two, all macro lines.

INSERT Command

Purpose: Inserts lines after a specified line until a null line is entered. Use INSERT to begin editing a new file or to insert lines in an existing file.

Format: I *after*
or
INSERT *after*

Remarks: *after* is the line after which the new lines are to be inserted.

To insert lines before the first line of a file, you specify 0 as the *after* parameter.

If you don't specify the *after* parameter, it defaults to the last line of the edit buffer.

To begin editing a new file, you use the INSERT command and don't specify the *after* parameter.

Trailing blanks (blank spaces at the end of lines) are not saved by the editor. Therefore, if trailing blanks are required, you enter them using the grave accent (`).

Example:

Allows you to insert lines after line 10. You are able to keep inserting lines until you enter a null line.

KEY Command

Purpose: Changes the value of the function keys.

Format:

```
K {string} keynumber
        or
KEY {string} keynumber
```

Remarks:

string	is a series of characters you want to assign to a key. If you don't supply the string originally, you are prompted for it.
keynumber	is the number of the function key you're changing.

If you want a key to be "hot" (execution immediately begins when you press the key), add the ¦ sign. For example, the F1 function key is a "hot" key. As soon as you press it, the HELP command is entered and executed.

Example:

```
K 6
```

Allows you to change the meaning assigned to F6.

Reference

The following prompt appears:

```
String:
```

You now type and enter the new definition you want to assign to function key F6.

```
LIST|
```

The LIST command now automatically executes when you press F6.

KILL
Command

Purpose: Erases a specified file from a diskette.

Format: KI *filespec*
or
KILL *filespec*

Remarks: *filespec* is the file specification of the file you want erased.

filespec defaults to the DOS drive (usually drive A).

Important: Since it destroys files, use the KILL command with caution. To make certain you want to erase a file, you are asked **KILL (Y/N)?** before a file is erased.

Example:
```
KI B:SAMPLE.SRC
```

Erases the file SAMPLE.SRC from the diskette in drive B.

KILL Command

You are asked to verify this command:

```
KILL (Y/N)?
```

If you decide to cancel this command, simply type **N** and press Enter. If you enter **Y**, the file is erased.

LIST Command

Purpose: Displays, on the screen, one or more lines of the edit buffer.

Format: L *first last width/Nonumber*

or

LIST *first last width/Nonumber*

Remarks:

first, last	are line numbers in the range of 1 to 999. *first* is the first line you're listing. *last* is the last line you're listing.
width	is the number of columns you're listing. The default is 255.
Nonumber	is an option to suppress line numbers. Specify it with the letter **N**.

width and *Nonumber* are mutually exclusive options. You can only use one of them.
If you specify the width as 0, you get the screen width minus five. This allows room for editor line numbers.
You can make the listing pause so you can look at your file by pressing the Space Bar. You also can cancel the LIST command by pressing the Enter key.

Example:

Lists lines 1 through 40 without editor line numbers.

Lists the first 20 columns of lines 1 through 40.

LOAD Command

Purpose: Loads a new file into the edit buffer for editing.

Format: LO *filespec*(.SRC) [*Append*] *first last*

or

LOAD *filespec*(.SRC) [*Append*] *first last*

Remarks:

filespec	is the file specification (device, filename, and extension) of the file you're loading. The default extension is .SRC.
Append	is an option to append a new file to the end of the file already in the edit buffer. Specify it with the letter **A**.
first, last	are line numbers in the range of 1 to 999. *first* is the first line of the specified file you're loading or appending. *last* is the last line of the specified file you're loading or appending. If *first* and *last* are not specified, the entire file is loaded.

If the *Append* option is not specified, all of the lines currently in the edit buffer are erased. For this reason, you are asked to verify the LOAD command if a file is already in the edit buffer.

Two of the function keys have been set to supply you with part of the LOAD COMMAND. You can use the F3 key to get **LOAD A:** or the F4 key to get **LOAD B:** and then supply the filename (and extension if necessary).

If you have a file without an extension, you must place a period after the filename to indicate that the file has no extension.

Example:
```
LO A:SAMPLE A 1 14
```

Appends lines 1 through 14 of the SAMPLE.SRC file to the end of the file already in the edit buffer.

You have to verify this command if lines are already in the edit buffer. This prompt appears on your screen:

```
Partial Appended LOAD (Y/N)?
```

Answer **Y** if you want lines 1 through 14 of the SAMPLE.SRC file appended to the file in the edit buffer.

Answer **N** if you don't want these lines appended.

MOVE Command

Purpose: Allows a single line or a group of lines to be moved after any existing line in the edit buffer. The editor lines then are renumbered.

Format: M *first last after*

or

MOVE *first last after*

Remarks:

first, last	are line numbers in the range of 1 to 999. *first* is the first line you're moving. *last* is the last line you're moving.
after	is the line after which the specified lines are moved.

To move only one line, specify it as both the first and the last lines. To move lines to the beginning of the file, specify that they be moved after line 0.

Note: You must enter all three parameters.

If the *Append* option is not specified, all of the lines currently in the edit buffer are erased. For this reason, you are asked to verify the LOAD command if a file is already in the edit buffer.

Unlike the COPY command, the MOVE command does not duplicate lines. Instead, the MOVE command places the lines in a new location and erases them from the old location.

Example: `4,7,14` [illegible]

Moves lines 4 through 7 after the current line 14 and deletes the original lines 4 through 7. All lines then are renumbered. Thus, lines 4 through 7 become the new lines 11 through 14.

NAME Command

Purpose: Changes the name of a diskette file.

Format: NA *oldfilespec* AS *newfilespec*
or
NAME *oldfilespec* AS *newfilespec*

Remarks: *oldfilespec* is the file specification of the file whose name you are changing.

newfilespec is the new filename and the device name, unless you want to use the default DOS drive.

The name you choose for the new filespec must not already exist on the designated diskette.

Example:

```
NA B:SAMPLE.SRC AS B:ASCII.SRC
```

Changes the name of the file SAMPLE.SRC on the diskette in drive B to ASCII.SRC, if this name does not already exist on the diskette.

NEW Command

Purpose: Clears the edit buffer.

Format: N
or
NEW

Remarks: You use the NEW command to start a different (or new) file. It clears the edit buffer, so you should save any work you want to preserve before using this command.

The NEW command has no parameters, since it simply clears the edit buffer.

Because this command can destroy material, you are asked to verify it. **Y** (Yes) clears the edit buffer; **N** (No) cancels the command without making any changes.

Example:

```
N
```

You get this prompt:

```
NEW (Y/N)?
```

PREP Command

Purpose: Converts a structured BASIC program to an executable BASIC program with actual BASIC language source statements.

Format: PRE [* /*filespec*(.BAS)] [* /*Nobackup*]
or
PREP [* /*filespec*(.BAS)] [* /*Nobackup*]

Remarks: * /*filespec* specifies the output file where the results of the preprocessing are saved. If * is specified, then the output file becomes the current filename with .BAS as the extension. If you chose to specify a filespec, .BAS is the default extension.

* /*Nobackup* specifies the backup file where the contents of the edit buffer are saved. If * is specified, then the current filespec becomes the backup file. If **N** is specified, then no back up is performed.

If nothing is specified, then BACKUP.TFE is used as a backup file.

Important: The PREP command destroys the current contents of the edit buffer, so using a backup parameter is recommended.

Example:

Preprocesses the file in the edit buffer and saves the results in a file with the same filename and .BAS as the extension. The source code is first saved in a backup file using the current filespec.

PRINT Command

Purpose: Prints, to the device you specify, one or more lines of the edit buffer.

Format: P [*filespec* [*Notitle*]] *first last width/Nonumber*

or

PRINT [*filespec* [*Notitle*]] *first last width/Nonumber*

Remarks:

filespec	is the file specification of the print destination. Enter it in the form of [*d*:] *filename*, where [*d*:] is the device and the *filename* includes the extension.

The filespec parameter is optional. The default for this is LPT1, which tells the program to print your file on the first printer connected to your computer. This printout contains date and time stamps. You also can print your file to a diskette by giving a filespec.

Notitle	is an option to suppress title lines. Specify it with the letter **N**.
first, last	are line numbers in the range of 1 to 999. *first* is the first line you're printing. *last* is the last line you're printing.

width is the number of columns you're printing.

If you specify the width as 0, you get the screen width minus five. The default for this parameter is 255.

Nonumber is an option to suppress line numbers. Specify it with the letter **N**.

width and *Nonumber* are mutally exclusive options. You only can use one of them.

Note: The F2 and F6 function keys have been assigned the PRINT command. If you press F2, you get **PRINT 0 0 N,** which gives you a printout of the entire file you have in the edit buffer without line numbers. You can change one or all of these parameters by using the Backspace key.

Example: `P 1 4 N`

Prints lines 1 through 4 without line numbers to LPT1: with title lines.

QUIT Command

Purpose: Allows you to leave the editor and return to DOS.

Format: Q
or
QUIT

Remarks: Because QUIT erases anything you have in the edit buffer, you are asked **QUIT (Y/N)?** before the command is executed. If you are certain you want to return to DOS, type **Y** and press the Enter key. If, however, you do not want to quit, type **N** and press Enter. The command then is cancelled.

Remember to save any work you want to preserve before using the QUIT command.

Example:

You now are asked to verify this command:

If you answer **Y**, this notice appears on your screen:

REPEAT Command

Purpose: Repeats a line or a group of lines.

Format: R *first last times*
or
REPEAT *first last times*

Remarks:

first, last	are line numbers in the range of 1 to 999. *first* is the first line you want repeated. *last* is the last line you want repeated.
times	is the number of times you want the lines repeated.

Unlike the COPY and MOVE commands, you cannot repeat lines anywhere but directly under the original lines.

With the REPEAT command, it is not necessary to supply all three parameters. If you enter only the first line, it is repeated once, directly under the original, and all of the lines in the edit buffer are renumbered. The default for the *times* parameter is 1.

Example:

```
R 6
```

Repeats line 6 as the new line 7 and renumbers all the following lines.

SAVE Command

Purpose: Saves the file in the edit buffer on the diskette.

Format: SA [*d:/filespec*(.SRC) [*Append*]]
first last

or

SAVE [*d:/filespec*(.SRC) [*Append*]]
first last

Remarks:

d:	is the drive to be used with the current filespec.
filespec	is the device, the filename, and the extension of the file you're saving. The default extension is .SRC.
Append	is the option to append the edit buffer to the end of the existing file.
first, last	are line numbers in the range of 1 to 999. *first* is the first line you're saving. *last* is the last line you're saving. If neither parameter is specified, the entire edit buffer is saved.

If you specify a filespec that differs from the filespec of the last file loaded, you are asked to verify this command. The question **SAVE (Y/N)?** appears on your screen.

You can save a file in several ways:

- If you are saving a new file to a diskette in the default drive, enter the filespec either by using the default extension .SRC or by supplying a different extension.
- If you are saving changes to an existing file using the same filespec, simply enter **SA**.
- If you are saving changes to the same filespec but on a different diskette drive, specify only the drive. For example, to save the current file to drive B, press F8 or type **SA B:** and press Enter.
- If you are saving the edit buffer to the diskette the file originally came from but giving the new file a new name, supply the filespec.
- If you are appending the edit buffer to another file, give the filespec for the file to which you are appending material and specify **A**.
- If you are appending the edit buffer to the original file, you still must give the filespec. *Append* cannot be used without a filespec.

SAVE Command

- If you want to save a file without an extension, you must specify the filename followed by a period. Otherwise, the default extension .SRC is used.

Example:

```
SA B:SAMPLE A
```

Appends the edit buffer to the SAMPLE.SRC file on the diskette in drive B.

SHIFT Command

Purpose: Moves one or more lines of the edit buffer a specified number of columns to either the left or the right.

Format: SH *first last amount*

or

SHIFT *first last amount*

Remarks:

first, last	are line numbers in the range of 1 to 999. *first* is the first line you're shifting. *last* is the last line you're shifting.
amount	is how far you want to move the lines and whether you want them moved to the right or to the left. A positive number shifts the lines to the right, and a negative number shifts the lines to the left. The default is 2, which shifts the lines two spaces to the right.

The SHIFT command works by padding the lines with blank spaces or by deleting characters, depending on whether you are shifting your file to the right or to the left.

Example: [illegible]

Shifts lines 1 through 10 five spaces to the right.

STATUS Command

Purpose: Displays the status of the editor, including the definitions of the function keys, the time, the amount of free space left in the edit buffer, and how many lines are in the edit buffer.

Format: S [*CLS*]
or
STATUS [*CLS*]

Remarks: *CLS* clears the screen, leaving only the command display and the command prompt visible.

Example: **S**

Displays the status of the file currently in the edit buffer.

TYPE Command

Purpose: Displays the contents of a specified diskette file on the screen, even if another file is in the edit buffer.

Format: T *filespec*(.SRC) *first last*
or
TYPE *filespec*(.SRC) *first last*

Remarks:

filespec	is the file specification of the file you want to display. The default extension is .SRC.
first, last	are line numbers in the range of 1 to 999. *first* is the first line you want to look at. *last* is the last line you want to look at. If neither parameter is specified, the entire file is displayed.

The TYPE command is comparable to the LIST command but with two differences. First, with the TYPE command you can display a file without destroying what you have in the edit buffer. Second, TYPE shows you the file without the editor line numbers.

Example:

```
T B:SAMPLE
```

Displays the SAMPLE.SRC file on the screen.

Appendixes

Contents

Appendix A. Messages

TFE/SBP

This section only lists the TFE/SBP messages. The FRMTBAS messages are in the second part of this appendix.

The TFE/SBP messages are listed alphabetically.

@ label undefined at line nnn

The specified label that you referenced in line nnn was never defined.

Add the missing label to your file and preprocess the file.

Buffer overflow

You have exceeded the limit of 999 lines in the edit buffer.

Break the file into smaller files to continue editing.

Buffer overflow at line nnn

During the UNPREP procedure, the number of lines reached exactly 999. All lines before nnn have been correctly structured and all lines after nnn are unstructured.

Break the program into smaller files and try to UN-PREP them.

Device not ready

The specified diskette drive may be open or empty. The printer may be out of paper or turned off.

Check to see that the disk drive has a diskette in it and that the door is closed. Also, check to see that your printer is turned on, is online, and has paper.

Disk full

You have insufficient space on the specified diskette to save your file.

Insert another diskette or erase some files and try the command again.

Disk media error

The system is having difficulty reading your diskette.

Insert your diskette again and repeat the command or the procedure. If it still doesn't work, you may have a bad diskette. Insert another diskette and try again.

Error xx in line nnn

This is a BASIC error message. The *xx* signifies the BASIC error message number, and the *nnn* signifies the line in which the error occurred.

Consult the IBM Personal Computer *BASIC* manual for information and instructions.

File already exists

You tried to rename a file to a name that already exists on that diskette.

Try the command using another filename.

File not found

The file you specified could not be found in the specified drive.

Check to see that you have the correct diskette in the drive, that you specified the correct drive, and that you typed the filespec correctly.

File not renumbered

The file you tried to UNPREP has not been properly renumbered.

See Appendix C for instructions on how to renumber your program.

File too large

Your file is too large for the edit buffer.

Break your file into smaller files to continue editing.

Invalid filespec

The device or filename is too long in the specified filespec.

Refer to the IBM Personal Computer *Disk Operating System* manual for information on filename specifications and try again.

Invalid syntax

You have incorrectly entered a command. The command name may be unknown, or the parameters may be missing, extraneous, or out of range.

Check the command syntax and try again.

Invalid syntax at line nnn

A structured macro parameter is missing or extraneous in line nnn.

Load the file again, correct the line, and preprocess the file.

Maximum nesting exceeded at line nnn

You have exceeded the limit of 10 levels of nested structured macros.

Replace the structure with a GOSUB to a new subroutine in which you may code another 10 levels of macro nesting.

No current filespec

You have attempted to preprocess a file that you have not saved.

Save the file and then preprocess it.

String too long

You have exceeded the limit of 255 characters on one line.

Break the line into two smaller lines to continue editing.

Structure mismatch at line nnn

During preprocessing, a structured macro was found to be out of sequence near the specified line.

Load the file again, correct the sequence of structured macros, and preprocess your file.

Too many files

You attempted to create a new file when the diskette is full.

Erase some files, or insert another diskette and try again.

Too many labels

You have exceeded the limit of 250 labels in one source file.

Break your file into smaller files that can be pre-processed separately to reduce the number of labels.

Diskette write-protected

The diskette in the specified drive is write-protected.

Remove the write-protect tab from the diskette, or insert another diskette.

FRMTBAS

This section only lists the FRMTBAS messages. Refer to the preceding pages for the TFE/SBP messages.

The FRMTBAS messages are listed in numerical order.

> **Note:** An * before the number means the error is a condition from which the utility cannot recover. Press Enter to return to the system. For all other errors, press Enter to continue.

Number	Message
1001	**Invalid filename** The filespec contains invalid characters or the filename has more than eight characters and the extension has been specified. Press Enter and type the correct filename or a suitable replacement.
1002	**Invalid drive** The drive specified is not supported. Press Enter and type in a supported drive.
1003	**Filename extension missing** The filename ended in a period, indicating, but not giving, a filename extension. Press Enter and type the correct filename, including the extension.

1004 Filename missing

Either the Enter key was pressed before the filename was typed, the filename was preceded by a period, or the drive was entered but followed by a semicolon rather than a colon.

Press Enter and type the correct filename.

1005 Input exceeds field length

You tried to enter more characters than the input field can contain, perhaps two characters for a one character prompt.

Press Enter and type in the correct character or characters.

***1006 Source file protected or not in ASCII format**

The file cannot be read by the BASIC formatter.

Press Enter to return to the system. Depending on the problem, either use an unprotected source file (if one is available), or convert the file to ASCII format.

1007 Non-numeric character in a numeric field

You tried to enter a non-numeric character for a numeric option selection.

Press Enter and type a valid option number.

1008 Input value is out of range

You tried to enter a numeric value that has no corresponding option.

Press Enter and type a valid option number.

***1009**	**Line overrun**
	The source program has too many nested FOR-NEXT or WHILE-WEND loops for the characters/line option.
	Press Enter to return to the system. Specify 132 characters per line and try again.
***1010**	**Statement reference table overflow**
	Your program contains too many GOTO, GOSUB, and/or "RESUME line" statements.
	Press Enter to return to the system. Split the program into two or more segments and then format each separately.
***1011**	**Variable name table overflow**
	Your source program contains too many variables, the variable names are too long, or you have selected too many reserved words.
	Press Enter to return to the system. Split the program into smaller modules or select fewer reserved words.
***1012**	**Variable reference table overflow**
	Your source program contains too many variable references and/or you have selected too many reserved word references.
	Press Enter to return to the system. Split the program into smaller modules or select fewer reserved words.
1013	**Printer option selected but no printer**
	A primary printer is not attached to the IBM Personal Computer.
	Attach a printer or select a different option.

1014 **File not found**

The designated source file does not exist on the specified (or default) drive.

Press Enter and type the filename again or verify that the correct diskette is in the correct drive.

1015 **Source file same as output file**

You have specified the same file for both input and output.

Press Enter to continue. Specify a different output file.

1016 **Invalid or repeated reserved words**

You have specified a word for reserved word references that is not in the reserved word list or have repeated a word.

Press Enter to continue. Check to see that the word is spelled correctly. If so, then you already have specified it.

***1017** **BIOS table error**

The program has interrogated the BIOS table to obtain various pieces of information, and one of the pieces is incorrect.

Turn off your system, wait at least 8 seconds, then turn it back on.

***1018** **Maximum nesting exceeded**

The number of nested BASIC structures (IF-THEN, FOR-NEXT, WHILE-WEND) has exceeded 35.

Press Enter to return to the system. You must change your program.

1114 Source file error nn

An error has occurred while reading the source file. This is a BASIC error message.

Refer to the IBM Personal Computer *BASIC* manual for the meaning of error code nnn.

***1115 Output file error nn**

An error has occurred while writing the output file to the diskette.

Refer to the IBM Personal Computer *BASIC* for the meaning of error code nn.

1116 File already exists

You have tried to direct output to a diskette file that already exists.

You can specify a different filename, press Enter to overwrite the existing file, or press the Escape (Esc) key to cancel.

1117 Ready drive

The drive is not ready.

Insure the drive door is properly closed and a diskette has been inserted.

1118 Diskette write-protected

The diskette in the specified drive is write-protected.

Insert an unprotected diskette for the output file, or remove the write-protect tab.

1119 Directory full

The output file diskette directory is full.

Insert another diskette or erase some files.

***1120 Disk full**

No more space is available on the diskette used for output. The file is closed at this point so that output is not lost. You return to the system if you have a one-drive machine or if the output is on the same drive as input.

Insert another diskette.

1121 Printer not available

The printer is not connected.

Check the connection and try again by pressing Enter.

1122 Printer out of paper or not turned on

The printer is out of paper or not turned on.

Check to see that there is paper in the printer and that the printer is turned on.

Appendix B. Converting Files to ASCII Format

This appendix shows you how to convert a BASIC "internal" file into ASCII format. If you're using the BASIC Formatter and Cross-Reference utilities, your file must be in ASCII format. Remember, if you created your file using the Text File Editor, it already is in ASCII format.

> **Note:** ASCII format requires more space than BASIC "internal" format. Allow extra space for this larger file.

To convert a file to ASCII format, follow these directions:

1. With your DOS diskette in the computer, enter BASIC by typing **BASIC** after the DOS prompt.
2. Load your program with **LOAD"filespec"**
3. Save your program with **SAVE"filespec",A**
4. Return to DOS by typing **SYSTEM** and pressing Enter.

Your file is now in ASCII format.

If you want to reverse the process, follow these steps:

1. With your DOS diskette in the computer, enter BASIC by typing **BASIC** after the DOS prompt.

2. Load your program with **LOAD"filespec"**

3. Save your program with **SAVE"filespec"**

Your file once again is stored in BASIC "internal" format.

Appendix C. Converting Existing Programs to Structured BASIC

The TFE/SBP has an EXEC file that takes existing BASIC programs and turns them into structured BASIC programs. This is the UNPREP EXEC file. For more information about EXEC files, refer to Lesson 1 in Chapter 4.

The following steps convert your program to structured BASIC.

> **Note:** *filespec* includes the filename and the extension.

1. Start BASIC. If you are in DOS, you have to enter the following command:

   Type **BASIC** and press Enter.

2. Load your program.

   Type **LOAD "[*d*:]*filespec*"** and press Enter.

3. Renumber your program.

   Type **RENUM 1,0,1** and press Enter.

   Note: You must enter the RENUM parameters as shown.

4. SAVE your program in ASCII format.

   Type **SAVE "[*d*:]*filespec*",A** and press Enter.

5. Return to DOS.

   Type **SYSTEM** and press Enter.

6. Start the Text File Editor and Structured BASIC Preprocessor.

   Type **TFE** and press Enter.

7. Load your source file (the one you saved in step 4).

   Type **LOAD[*d*:]*filespec*** and press Enter.

8. Execute the UNPREP procedure.

   Type **EX UNPREP** and press Enter.

When the UNPREP procedure completes, the edit buffer contains a structured BASIC program, including labels and the appropriate macros. Save this structured BASIC program.

> **Note:** You must save your file after UNPREP if you are going to use the PREP command on it. The PREP command only works after a SAVE command or a LOAD command.

Appendix D. Other Ways to Preprocess

DEBUG

The DEBUG EXEC file is another way of preprocessing structured code. It expands the macros, turns the labels into line numbers, and creates an output file. This output file is logically equivalent to the one PREP creates, except that the lines have not been compressed. The preprocessed code is saved to the current filespec and the extension .BAS is added.

Unlike PREP, DEBUG retains the remarks by turning them into REM statements and does not combine lines. For this reason, DEBUG produces a more readable version of your code for testing and debugging. The code, however, requires more space and probably won't run as fast as the code PREP produces.

The DEBUG procedure automatically includes all code following pairs of single quotes (' '). During a normal PREP, these lines are treated as remarks, and the entire line is deleted. In DEBUG, however, a single quote (') is changed into REM and a pair of single quotes (' ') is deleted, making the rest of the line executable.

This allows you to code special routines that appear only in test versions of your programs.

To use the DEBUG EXEC file, you type **EX DEBUG** and press Enter.

> **Important:** DEBUG does not back-up your source.

FREQ

The FREQ EXEC file preprocesses structured BASIC code by expanding the macros, changing the labels into line numbers, creating an output file (*filename*.BAS), and deleting remarks if they're specified by a single quote (').

In addition, the FREQ procedure adds code to your progam that upon execution creates a frequency file, TEMP.TFE. This file contains the number of times each line of the program was executed.

To use the FREQ EXEC file, you type **EX FREQ** and press Enter.

Important: FREQ does not back up your source.

Appendix E. Editing and Preprocessing Large Programs

Editing

To edit a program that is over 45KB or 999 lines, you break it into manageable pieces, edit the pieces, and then put the pieces back together.

You use the partial load option of the LOAD command to load a piece of your program into the edit buffer. When you've finished editing, you save the edited piece to a new file and append the other pieces to it.

For example, assume you need to edit a program that is 2100 lines long. You use the following sequence of commands:

Steps	Operation
Load first 700 lines Verify partial LOAD	Enter command: LOAD BIGFILE 1 700 Partial LOAD (Y/N)? Y
Perform editing	
Save first 700 lines Verify SAVE to different file	Enter command: SAVE SMLFILE SAVE (Y/N)? Y
Load second 700 lines Verify partial LOAD	Enter command: LOAD BIGFILE 701 1400 Partial LOAD (Y/N)? Y
Perform editing	
Save second 700 lines Verify appended SAVE	Enter command: SAVE SMLFILE APPEND Appended SAVE (Y/N)? Y
Load last 700 lines Verify partial LOAD	Enter command: LOAD BIGFILE 1401 Partial LOAD (Y/N)? Y
Perform editing	
Save last 700 lines Verify appended SAVE	Enter command: SAVE SMLFILE APPEND Appended SAVE (Y/N)? Y

Preprocessing

The same limitations that apply to editing a large program also apply to preprocessing a large program. The solution to this problem is the same, that is, split the program into pieces, preprocess the pieces individually, and then merge the pieces together.

This solution, however, presents problems for the preprocessor. For example, the first piece of a program may include label references contained in the other pieces of the program.

When you attempt to preprocess the first piece of the program, the preprocessor encounters these references for which it cannot find the corresponding label and preprocessing fails. Therefore, the following solution is suggested.

Unless otherwise specified, all structured BASIC programs begin with line number 1 and increment by 1. You may change this by specifying a positive number as the first string of non-blank characters on the line. This should precede any labels that appear on the line and must be followed by at least one blank.

If only a label appears following the line number, no code is generated, but the label is assigned that value during preprocessing. This and all following lines then are numbered starting with the specified number and incrementing by 1. This provides the ability to include line numbers in one file of the program that are found in other files of the program.

The following is an example of a multiple file program in which the main program resides in one file and the subroutines reside in another file.

Main program file, MAIN.SCR:

```
1000 @SUB 1 'establish line number of first subroutine
2000 @SUB2 'establish line number of second subroutine
1 @MAIN      'establish first line number of main program
PRINT "START OF MAIN PROGRAM"
GOSUB @SUB1
PRINT "MIDDLE OF MAIN PROGRAM"
GOSUB @SUB2
PRINT "END OF MAIN PROGRAM"
END
```

Subroutine file, SUBS.SRC:

```
1000 @SUB1 'establish first line number of first subroutine
PRINT "FIRST SUBROUTINE"
RETURN
2000 @SUB2 'establish first line number of second subroutine
PRINT "SECOND SUBROUTINE"
RETURN
```

These files are preprocessed separately and merged back together as shown on the next page.

Steps	Operation
Load main program file	Enter command: LOAD MAIN
Verify Load	LOAD (Y/N)? Y
(no prompt if first time)	
Preprocess main program	Enter command: PREP
Verify preprocess command	PREP (Y/N)? Y
Load subroutines file	Enter command: LOAD SUBS
Verify LOAD	LOAD (Y/N)? Y
Preprocess subroutine programs	Enter command: PREP
Verify preprocess command	PREP (Y/N)? Y
Exit TFE/SBP	Enter command: QUIT
Verify QUIT	QUIT (Y/N)? Y
Load BASIC	A > BASIC Ok
Load main program	LOAD"MAIN Ok
Merge subroutines with main program	MERGE"SUBS Ok
Save combined programs	SAVE"PROGRAM Ok
Run program	RUN
·	START OF MAIN PROGRAM
·	FIRST SUBROUTINE
· (output of program)	MIDDLE OF MAIN PROGRAM
·	SECOND SUBROUTINE
·	END OF MAIN PROGRAM
	Ok

Appendix F. Editing an EXEC File

When you enter TFE/SBP from DOS, an EXEC file, called AUTOEXEC.TFE, automatically executes to assign default values to the function keys.

This EXEC file, like any EXEC file, may be edited with the Text File Editor. Thus, it is possible to change the default values associated with any of the function keys simply by editing AUTOEXEC.TFE.

First, load AUTOEXEC.TFE and list it.

Type **LOAD A:AUTOEXEC.TFE** and press Enter.

Type **L** and press Enter.

The file looks like this:

```
001:KEY 1
002:HELP¦
003:KEY 2
004:PRINT 0 0 N
005:KEY 3
006:LOAD A:
007:KEY 4
008:LOAD B:
009:KEY 5
010:GOSUB @
011:KEY 6
012:PRINT
013:KEY 7
014:SAVE A:
015:KEY 8
016:SAVE B:
017:KEY 9
018:FILES A:*.*¦
019:KEY10
020:FILES B:*.*¦
021:STATUS CLS
```

Because both F2 and F6 are set for the PRINT command, let's change F6 to LIST. You follow these steps:

1. Use the CHANGE command to assign LIST to F6. Type **C 12 12** and press Enter. (The number 12 specifies that you want to change lines, beginning with and ending with line 12.)

2. You're prompted for the strings. Type **PRINT** in response to the first prompt, **OLD:**, and press Enter. Type **LIST¦** in response to the second prompt, **NEW:**, and press Enter. The ¦ sign indicates that you want LIST to be a "hot" key.

   Your screen looks like this:

   ```
   OLD: PRINT
   NEW: LIST¦
   ```

3. Save the changed file in AUTOEXEC.TFE.

   Type **SAVE** and press Enter.

At this point, you've only changed the file responsible for assigning default values to the function keys. When TFE/SBP is exited and subsequently started, then the value of F6 is assigned the default value of LIST¦.

If you want to reassign F6 only for your current session, then you must use the KEY command.

Index

Continued from inside front cover

SOME STATES DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO THE ABOVE EXCLUSION MAY NOT APPLY TO YOU. THIS WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS AND YOU MAY ALSO HAVE OTHER RIGHTS WHICH VARY FROM STATE TO STATE.

IBM does not warrant that the functions contained in the program will meet your requirements or that the operation of the program will be uninterrupted or error free.

However, IBM warrants the diskette(s) or cassette(s) on which the program is furnished, to be free from defects in materials and workmanship under normal use for a period of ninety (90) days from the date of delivery to you as evidenced by a copy of your receipt.

LIMITATIONS OF REMEDIES

IBM's entire liability and your exclusive remedy shall be:

1. the replacement of any diskette(s) or cassette(s) not meeting IBM's "Limited Warranty" and which is returned to IBM or an authorized IBM PERSONAL COMPUTER dealer with a copy of your receipt, or
2. if IBM or the dealer is unable to deliver a replacement diskette(s) or cassette(s) which is free of defects in materials or workmanship, you may terminate this Agreement by returning the program and your money will be refunded.

IN NO EVENT WILL IBM BE LIABLE TO YOU FOR ANY DAMAGES, INCLUDING ANY LOST PROFITS, LOST SAVINGS OR OTHER INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE SUCH PROGRAM EVEN IF IBM OR AN AUTHORIZED IBM PERSONAL COMPUTER DEALER HAS BEEN ADVISED OF THE POSSIBLITY OF SUCH DAMAGES, OR FOR ANY CLAIM BY ANY OTHER PARTY.

SOME STATES DO NOT ALLOW THE LIMITATION OR EXCLUSION OF LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES SO THE ABOVE LIMITATION OR EXCLUSION MAY NOT APPLY TO YOU.

GENERAL

You may not sublicense, assign or transfer the license or the program except as expressly provided in this Agreement. Any attempt otherwise to sublicense, assign or transfer any of the rights, duties or obligations hereunder is void.

This Agreement will be governed by the laws of the State of Florida.

Should you have any questions concerning this Agreement, you may contact IBM by writing to IBM Personal Computer, Sales and Service, P.O. Box 1328-W, Boca Raton, Florida 33432.

YOU ACKNOWLEDGE THAT YOU HAVE READ THIS AGREEMENT, UNDERSTAND IT AND AGREE TO BE BOUND BY ITS TERMS AND CONDITIONS. YOU FURTHER AGREE THAT IT IS THE COMPLETE AND EXCLUSIVE STATEMENT OF THE AGREEMENT BETWEEN US WHICH SUPERSEDES ANY PROPOSAL OR PRIOR AGREEMENT, ORAL OR WRITTEN, AND ANY OTHER COMMUNICATIONS BETWEEN US RELATING TO THE SUBJECT MATTER OF THIS AGREEMENT.

International Business Machines Corporation

P.O. Box 1328-C
Boca Raton, Florida 33432

6936756
Printed in United States of America